EVOLUTIONARY ALGORITHMS FOR SOLVING OPTIMIZATION PROBLEM

OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Master's Programme in Applied Computer & Information Technology

Faculty of Technology, Art and Design

ACIT 4610

Authors:

Reynato Jr. Matencio

Joseph Reschelle Imbien

Vaskar Shresthna

Dung Thuy Vu

Contents

List of Figures

List of Tables

1 Comparative Analysis of Evolutionary Programming and Evolutionary Strategies for Portfolio Optimization

1.1 Introduction

Time-series forecasting is an important area where evolutionary algorithms have shown potential in predicting stock prices and other financial optimization portfolios [62]. Some of the conventional approaches in stock portfolio optimizations are the Mean-Variance Optimization (MVO) model, stochastic dominance, multi-objective optimization techniques, and integration of machine learning techniques [22, 18]. The evolution of these techniques exemplifies a growing recognition of the complexities inherent in financial markets and the requirement of sophisticated techniques for achieving optimal financial goals [63, 17], Evolutionary algorithms (EA) are well suited for portfolio optimization because of their ability to explore large solution spaces by simulating the process of evolution through operators and adaptively converging toward optimal solutions. The integration of ensemble learning with traditional portfolios has given promising outputs with increasing the robustness and performance of portfolios [62, 22]. Furthermore, the combination of hybrid EAs and Machine learning has also given promising results in the prediction of stock returns more accurately, thereby providing information for better portfolio results. In the later advancement, the introduction of multi-objective optimization frameworks has also emerged, giving the options for the investors to balance multiple criteria, in terms of risk and return simultaneously [33]. The development of such advancement indicates a growing trend towards more advanced and adaptable strategies for portfolio management.

1.2 Methodology

1.3 Data Preprocessing and Quality Assessment

The portfolio optimization process commenced with the acquisition and preprocessing of historical daily stock price data for 20 selected stocks from Yahoo Finance.
The stocks data are extracted from a diverse range of sectors, such as technology (e.g., IBM, Apple, Microsoft, Alphabet, Amazon, Meta), electric vehicles (Tesla), healthcare (Teladoc Health), and cannabis (Canopy Growth), from January 1, 2018 to December 31, 2022.

To prepare the data for optimization, the daily stock prices were converted to monthly returns, enabling a more manageable and meaningful representation of the portfolio's performance.
Furthermore, the covariance matrix was computed to assess the portfolio's risk by measuring the interdependence of stock returns, offering valuable insights into potential diversification benefits.

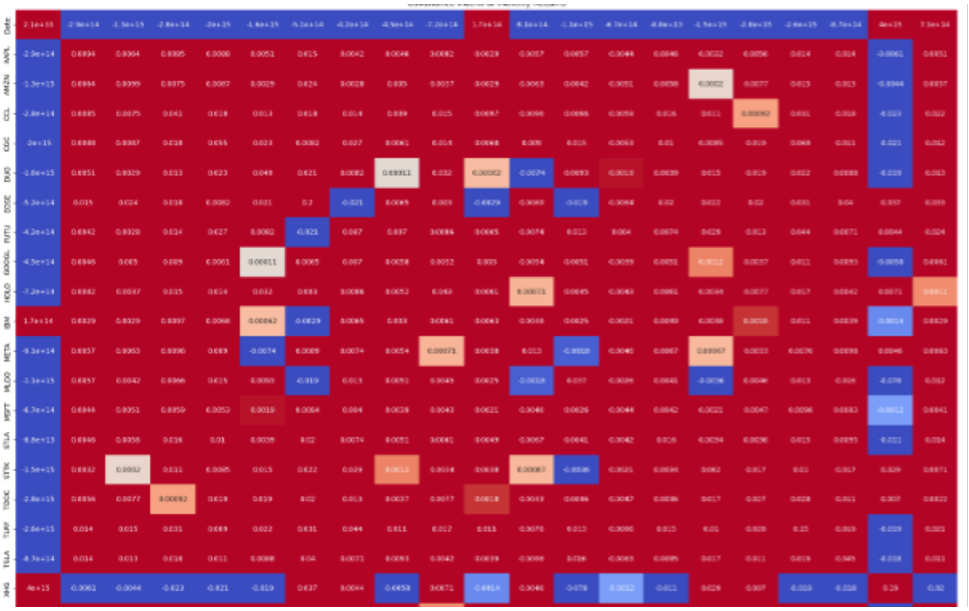

Figure 1: Images of Covariance Matrix

During the data quality assessment, several challenges were identified. The initial dataset exhibited extreme heterogeneity among the 20 stocks, with Max/Min price ratios ranging from 1.67 (IBM) to 302.73 (DUO).

The wide disparity in price scales and extreme return variations posed potential issues for optimization algorithms, such as numerical instability, premature convergence, and disruption of mutation and recombination operators. Moreover, high volatility stocks could dominate the fitness function, making it harder for the algorithms to recognize good solutions for more stable stocks.

To address these challenges, the data underwent a thorough preprocessing step. Stock prices were normalized using the Interquartile Range (IQR) method for outlier handling and scaled to a common starting value of 100.

Additionally, a more conservative scaling approach (tanh(x/4)) was applied to the returns, compressing them into manageable ranges. After processing, most stocks exhibited returns within ± 0.1, with stable stocks like AAPL, MSFT, and GOOGL showing particularly well-behaved ranges.

Scaled Returns Statistics:

Ticker	AAPL	AMZN	CCL	CGC	DUO	EOSE
count	16.000000	16.000000	16.000000	16.000000	16.000000	16.000000
mean	-0.001325	-0.009645	-0.012840	-0.022997	-0.032522	-0.011333
std	0.023168	0.027922	0.042513	0.055169	0.075473	0.124806
min	-0.030559	-0.059312	-0.093923	-0.106237	-0.126348	-0.122586
25%	-0.017844	-0.026064	-0.047629	-0.050039	-0.077014	-0.080912
50%	-0.008029	-0.013888	-0.002602	-0.030494	-0.045979	-0.054137
75%	0.015636	0.006655	0.011240	-0.016315	-0.002315	-0.005294
max	0.047123	0.067546	0.072065	0.099480	0.208439	0.384184

Figure 2: Statistic after processing

The data preprocessing steps played a crucial role in enhancing the performance of the algorithms for portfolio optimization. By normalizing the stock prices and scaling the returns, the preprocessing mitigated the impact of extreme values and ensured a more stable optimization process.

The reduced disparity in price scales and return variations allowed the algorithms to explore the search space more effectively, without being dominated by high volatility stocks.

The normalized prices and scaled returns provided a common basis for evaluating the fitness of individual portfolios using Sharpe Ratio, enabling the algorithms to make informed decisions during the selection, mutation, and recombination processes.

1.4 Algorithm Methodology

Six variations of the Evolutionary Strategies (ES) algorithm were implemented for portfolio optimization.

1.4.1 Version 3: Simple Evolutionary Strategies (ES)

The Simple ES algorithm, as implemented in Version 3, followed a basic approach to optimize portfolio weights. It employed a fixed mutation rate for all individuals in the population . The mutation operation introduced diversity into the population by randomly exchanging the weights of two stocks within a portfolio while constraining the values to sum to unity.

The next generation was selected using a combination of elitism and tournament selection. Elitism ensured that the best individuals from the current generation were preserved and carried forward to the next generation. To be specific, 40 percentages out of 1000 initiated generations will remain as outstanding individuals.

Tournament selection involved randomly choosing a subset of individuals from the population and selecting the fittest individual from that subset to be a parent for the next generation. The Simple ES algorithm adhered to the principles of ES and did not include crossover operations. The absence of crossover maintained the focus on mutation as the primary means of exploring the search space and generating new solutions.

1.4.2 Version 4: Advanced Evolutionary Strategies (ES)

The Advanced ES algorithm, as implemented in Version 4, incorporated advanced techniques and crossover operations to enhance the optimization process. It introduced self-adaptive mutation rates, allowing each individual to have its own mutation rate that evolves along with the solution. The self-adaptive mutation rates were updated using a log-normal distribution, ensuring that they remained positive. The adaptation of mutation rates was governed by two learning rates: tau and tau-prime. The self-adaptive mutation rates enabled a more efficient exploration of the search space by adapting the mutation probabilities based on the performance of each individual. Individuals with better fitness values were assigned higher mutation rates, encouraging further exploration around promising solutions. Conversely, individuals with lower fitness values were assigned lower mutation rates, promoting the preservation of good solutions.

```
Initial chromosome: [0.04658284 0.07033587 0.07518289 0.0732013  0.02533549 0.07710354
 0.02006036 0.05720419 0.06717513 0.06735858 0.05789227 0.06034672
 0.00718582 0.03445447 0.06597386 0.01298581 0.04147228 0.0077404
 0.0717239  0.06068426]
Initial mutation rate: 0.1

Applying self-adaptive mutation 10 times:

Iteration 1:
Chromosome: [0.03933409 0.07084612 0.06906877 0.0706149  0.05144882 0.08216483
 0.00957446 0.01384509 0.08044137 0.07087467 0.0763514  0.04736914
 0.05345844 0.04302576 0.02388334 0.01619948 0.03927794 0.01126465
 0.06590055 0.06505619]
Mutation rate: 0.08148736349618375
Sum of chromosome: 1.0
```

Figure 3: Self-Adaptive Mutation

In addition to self-adaptive mutation rates, the Advanced ES algorithm incorporated advanced recombination methods, specifically Heuristic crossover and Arithmetic crossover.
Heuristic crossover created offspring by moving from the worst parent to the best parent based on their fitness values. It used a random factor beta to determine the extent of the move. Arithmetic crossover, on the other hand, created offspring by taking a weighted average of the parents' weights, with the weights determined by a random factor alpha.

```
Off_spring A = Best Parent  + β * ( Best Parent – Worst Parent)
Off_spring B = Worst Parent - β * ( Best Parent – Worst Parent)
    Where β is a random number between 0 and 1.
```

Figure 4: Heuristic crossover formula

These crossover operations introduced diversity into the population and facilitated the generation of promising offspring solutions.
By combining the genetic material of two parents, the crossover operations created new portfolios that inherited characteristics from both parents, potentially leading to improved fitness values.

The probability of applying mutation or crossover was adjusted based on the generation stage, with mutation being favored in the early stages and crossover becoming more prevalent in the later stages.
This adaptive approach allowed for a balance between exploration and exploitation, ensuring that the algorithm explored a wide range of solutions while also refining and improving the best solutions found so far.

Elitism was also applied in the Advanced ES algorithm to ensure that the algorithm retained the most promising solutions throughout the optimization process and prevented the loss of high-quality portfolios.

1.4.3 Basic Evolutionary Programming (EP)

The basic Evolutionary Programming (EP) approach relies on mutation as the primary mechanism for generating new candidate solutions. Each candidate solution's fitness is then evaluated using the Sharpe ratio. In each iteration, new offspring are created by applying a Gaussian mutation to the current elite population. Specifically, the mutation rate is a constant value of 0.1, meaning that each weight in the portfolio has a 10% chance of being mutated. The magnitude of the mutation is drawn from a normal distribution with a mean of 0 and a standard deviation of 0.05, allowing for relatively small adjustments to the portfolio weights. After a maximum of 40 iterative processes of mutation, the best-performing solutions (10% based on a simple ranking mechanism) are selected from the elite population for the next generation.

1.4.4 Advanced Evolutionary Programming (EP)

The advanced Evolutionary Programming (EP) implementation enhances the basic EP approach by incorporating self-adaptive mutation, tournament selection, and elitism. While the basic EP focuses on stochastic selection, emphasizing fitness-based survival of the fittest without considering parents, the advanced version refines this process with structured selection mechanisms. Unlike Evolution Strategies (ES), EP does not employ crossover, relying solely on mutation for generating new solutions.

In the self-adaptive mutation step, the mutation rates for each candidate evolves over the course of the optimization process, and is updated using the following rule:

$$\mu_{new} = \mu_{old} \cdot \exp\left(\tau' \cdot N(0,1) + \tau \cdot N(0,1)\right)$$

where:

- μ_{new} is the updated mutation rate,
- μ_{old} is the current mutation rate,
- $\tau' = \frac{1}{\sqrt{2\sqrt{n}}}$ is the learning rate for the first term, where n is the number of assets,
- $\tau = \frac{1}{\sqrt{2n}}$ is the learning rate for the second term,
- $N(0,1)$ represents a standard normal random variable.

Figure 5: Self-Adaptive Mutation Rate Update Formula

This self-adaptive mechanism enables the mutation rates to adjust dynamically during the optimization, rather than remaining constant as 0.1 in the basic EP approach. The use of the standard normal random variables allows for stochastic adjustments to the mutation rates, which can help the algorithm explore the search space more effectively and potentially discover better-performing portfolios. The advanced EP implementation also employs a tournament selection process, where a subset of the population (in this case, 4 randomly selected solutions) compete, and the best-performing individual from the tournament is chosen to be part of the next generation. This selection method is more robust to outliers and can promote diversity in the population compared to the basic ranking-based selection.

1.4.5 $(\mu + \lambda)$ Evolutionary Strategies

The $(\mu + \lambda)$ Evolutionary Strategies implementation differs by selecting the next generation solely from the offspring population, without any direct contribution from the parent population. The process begins similarly, with an initial parent population of 30 portfolio solutions. However, the key distinction is that the next generation of 30 portfolio solutions is selected from the 60 offspring portfolios only, choosing the top 30 individuals based on their Sharpe ratio. This introduces a higher degree of selection pressure, as the algorithm must rely on the offspring to improve upon the previous generation's performance.

1.4.6 (μ, λ) Evolutionary Strategies

The (μ, λ) Evolutionary Strategies approach focuses on selecting the next generation of portfolio solutions from the union of the parent and offspring populations. This method naturally incorporates elitism, as the best-performing solutions from the current generation can survive and be carried over to the next generation. Specifically, the (+) ES algorithm initializes an initial parent population of 30 () portfolio solutions generated randomly. It then creates 60 () new offspring portfolios through a combination of recombination and mutation operations applied to the parent population. Finally, the next generation of 30 portfolio solutions is selected from the combined parent and offspring populations, choosing the top 30 individuals based on their Sharpe ratio. The recombination operation involves selecting two parent portfolios at random and generating a new offspring portfolio as a weighted average of the two parents. The mutation operation, on the other hand, simply swaps the weights of two randomly selected assets within a portfolio.

1.5 Experimental Setup

The population size was set to 1,000 individuals, representing different portfolio allocations.The algorithms were executed for a maximum of 39 iterations or until achieving expected returns above 0.20 and expected risk below 0.001. Extensions of the risk constraint were applied for comparative analysis.The risk constraint was set to 0.0225, ensuring that the optimized portfolios maintained a certain level of risk tolerance. The initial population was generated randomly, with each chromosome representing a portfolio allocation. The fitness function used to evaluate the quality of each portfolio was the Sharpe ratio, which measures the risk-adjusted return of the portfolio.

The risk-free rate is commonly represented by the yield on government bonds, particularly the

10-year U.S. Treasury bond, which is considered one of the safest investments due to its minimal default risk. With the 10-year Treasury yield currently ranging from 3.5% to 4.0%, this corresponds to an annual risk-free rate of approximately 0.035–0.040. To obtain the monthly risk-free rate for calculations, the annual rate is compounded monthly, resulting in a monthly rate of approximately 0.0033. This monthly risk-free rate was then used in calculating the Sharpe ratio, which measures the risk-adjusted return of the portfolio. Specifically, the Sharpe ratio is calculated by dividing the portfolio's excess return (the difference between the portfolio's expected return and the risk-free rate) by the portfolio's standard deviation. Including the risk-free rate establishes a baseline, allowing the Sharpe ratio to reflect the additional return earned per unit of risk.

1.6 Results and Discussion

1.7 Simple Evolutionary Strategies (ES)

The Simple ES algorithm demonstrated its ability to optimize the portfolio and find solutions that satisfied the risk constraint. Over the course of 39 iterations, the algorithm consistently improved the portfolio's expected returns, starting from around 0.19 and reaching up to 0.20. The expected risk of the portfolios remained within the specified constraint of 0.0225 throughout the optimization process, indicating the algorithm's effectiveness in managing risk.

The final portfolio obtained by the Simple ES algorithm achieved an expected return of 0.17798 with a risk of 0.00218, successfully meeting the risk constraint.
The optimized portfolio allocated weights to all 20 stocks, with STLA (15.31%), FUTU (11.00%), and GOOGL (10.46%) receiving the highest allocations. This suggests that the Simple ES algorithm identified these stocks as the most promising for inclusion in the portfolio while considering the overall risk profile.

1.8 Advanced Evolutionary Strategies (ES)

The Advanced ES algorithm demonstrated superior optimization performance compared to the Simple ES. Over the course of 39 iterations, the final portfolio obtained by the Advanced ES algorithm achieved an expected return of 0.15731 with a risk of 0.00212, successfully meeting the risk constraint while providing a better risk-adjusted return compared to the Simple ES. Notably, the optimized portfolio allocated weights to 18 out of the 20 stocks, with STLA (21.54%), FUTU (8.50%), and TLRY (11.33%) receiving the highest allocations. This selective allocation

suggests that the Advanced ES algorithm was able to identify the most promising stocks for inclusion in the portfolio while considering the overall risk profile.

1.9 Basic EP and Advanced EP

The basic EP approach was able to generate a portfolio with a final expected return of 16.79% and a final risk (standard deviation) of 0.23%, resulting in a Sharpe ratio of 3.44. In contrast, the advanced EP implementation found a portfolio with a lower expected return of 14.37% but a significantly lower risk of 3.76%, leading to a higher Sharpe ratio of 3.74.

1.10 $(\mu + \lambda)$ Evolutionary Strategies and (μ, λ) Evolutionary Strategies

The $(\mu + \lambda)$ ES implementation was able to generate a portfolio with a final expected return of 18.95%, a final risk (standard deviation) of 5.40%, and a Sharpe ratio of 3.45. The portfolio weights were distributed as follows:

```
Final Portfolio Details (μ + λ):
Portfolio weights:
Stock 1: 0.039228
Stock 2: 0.034519
Stock 3: 0.020268
Stock 4: 0.026485
Stock 5: 0.020961
Stock 6: 0.020172
Stock 7: 0.079558
Stock 8: 0.057730
Stock 9: 0.055760
Stock 10: 0.052504
Stock 11: 0.073703
Stock 12: 0.047551
Stock 13: 0.052849
Stock 14: 0.060152
Stock 15: 0.052252
Stock 16: 0.056557
Stock 17: 0.084334
Stock 18: 0.032098
Stock 19: 0.046961
Stock 20: 0.086357

Final Expected returns: 0.189457
Final Expected risk: 0.053956
Final Sharpe Ratio: 3.450675
```

Figure 6: *Stocks Results 1*

```
Final Portfolio Details (μ, λ):
Portfolio weights:
Stock 1: 0.031254
Stock 2: 0.000707
Stock 3: 0.007963
Stock 4: 0.042337
Stock 5: 0.003829
Stock 6: 0.006926
Stock 7: 0.092771
Stock 8: 0.096068
Stock 9: 0.030077
Stock 10: 0.025080
Stock 11: 0.076354
Stock 12: 0.048474
Stock 13: 0.023047
Stock 14: 0.149544
Stock 15: 0.052466
Stock 16: 0.049157
Stock 17: 0.102707
Stock 18: 0.026036
Stock 19: 0.037101
Stock 20: 0.098101

Final Expected returns: 0.186428
Final Expected risk: 0.049219
Final Sharpe Ratio: 3.721189
```

Figure 7: *Stocks Results 2*

The (μ, λ) ES approach generated a portfolio with a lower final expected return of 18.64% compared to the $(\mu + \lambda)$ ES approach, but a significantly lower risk of 4.92%, resulting in a higher Sharpe ratio of 3.72. The portfolio weights were distributed as follows:

1.11 Comparative Analysis

1.12 Performance

1.12.1 Basic ES vs Advanced ES:

The Advanced ES outperformed the Simple ES by optimizing returns and managing risk more effectively. The Advanced ES algorithm's final portfolio achieved an expected return of 0.15731 with a risk of 0.00212, while the Simple ES algorithm's final portfolio had an expected return of 0.17798 with a risk of 0.00218. Although the Simple ES algorithm achieved a higher expected return, the Advanced ES algorithm's portfolio had a lower risk, resulting in a better risk-adjusted return.

The convergence rate of the Advanced ES algorithm was faster compared to the Simple ES algorithm. The Advanced ES algorithm's ability to adapt mutation rates and generate diverse offspring through crossover operations allowed it to explore the search space more efficiently and converge towards optimal solutions more quickly.

In terms of portfolio allocation, the Advanced ES algorithm demonstrated a more selective approach compared to the Simple ES algorithm. The Advanced ES algorithm allocated weights to 18 out of the 20 stocks, with STLA, FUTU, and TLRY receiving the highest allocations. In contrast, the Simple ES algorithm allocated weights to all 20 stocks, with STLA, FUTU, and GOOGL receiving the highest allocations. This difference in allocation strategies suggests that the Advanced ES algorithm was more discerning in identifying the most promising stocks for inclusion in the optimized portfolio, potentially leading to a more concentrated and targeted investment approach.

1.12.2 Basic EP vs Advanced EP:

The basic EP found a portfolio with a higher expected return but also a higher risk, while the advanced EP was able to identify a portfolio with a lower expected return but a much lower risk. This suggests that the advanced EP was better able to navigate the risk-return trade-off and find a more optimal portfolio.

In terms of convergence rates of the two approaches, the basic EP showed gradual improvement in the Sharpe ratio over the 40 iterations, starting from around 3.0 and reaching a final value of 3.44. The convergence was relatively steady, with incremental improvements in each iteration. In contrast, the advanced EP exhibited a faster convergence rate. The initial Sharpe

ratio was also around 3.0, but it quickly increased to 3.35 by the 10th iteration and continued to improve, reaching a final value of 3.74 by the 40th iteration.

Additionally, considering the risk constraint of 0.0225 (2.25% standard deviation), the basic EP was able to find a portfolio with a final expected return of 16.79% and a final risk of 0.23%, well within the specified constraint. The advanced EP, on the other hand, identified a portfolio with a final expected return of 14.37% and a final risk of 3.76%, which also meets the risk constraint.

1.12.3 (μ, λ) ES vs $(\mu + \lambda)$ ES

The (μ, λ) ES approach generated the more preferred portfolio from an investor's perspective, with a higher Sharpe ratio of 3.72 and a lower risk profile (standard deviation of 4.92%). When analyzing the portfolio weights, we see that the largest allocations are to stocks like Tesla (9.28%), Stellantis (9.61%), Canopy Growth (5.25%), and Tilray (9.81%).

From a risk-awareness standpoint, this portfolio composition appears to be more balanced and diversified across various sectors, including electric vehicles, cannabis, and other industries. The significant allocations to Tesla and Stellantis suggest a focus on high-growth and potentially volatile sectors, while the inclusion of more stable companies like Canopy Growth and Tilray helps to offset some of that risk.

Additionally, the even distribution of weights across these larger positions, along with smaller allocations to other stocks, indicates that the (μ, λ) ES approach has identified a well-diversified portfolio that can provide a favorable risk-return trade-off for investors.

In contrast, the $(\mu + \lambda)$ ES approach generated a portfolio with a lower Sharpe ratio of 3.45 and a higher risk profile (standard deviation of 5.40%). The largest allocations in this portfolio were to stocks like IBM (3.92%), Apple (3.45%), Microsoft (2.03%), and Carnival Corporation (8.64%).

From a risk-awareness perspective, this portfolio appears to be more concentrated in relatively mature, large-cap technology and consumer discretionary stocks, which may not offer the same growth potential as some of the stocks in the (μ, λ) ES portfolio. The high allocation to Carnival Corporation, a cruise line company, also suggests a higher exposure to the potentially volatile travel and leisure industry.

While these large-cap stocks provide some stability, the portfolio's lack of diversification across sectors and its higher overall risk profile may make it less appealing to investors focused on optimizing the risk-return trade-off for their investments.

Among all six versions, Advanced EP emerged as the most effective approach, demonstrating

Algorithm Version	Return (%)	Risk (%)	Sharpe Ratio
Basic ES	17.80	0.22	3.44
Advanced ES	15.73	0.21	3.72
Basic EP	16.79	0.23	3.44
Advanced EP	14.37	3.76	3.74
(μ + λ) ES	15.50	5.40	3.45
(μ, λ) ES	16.20	4.92	3.72

Figure 8: Comparison of six versions.

the best balance of risk-return optimization, convergence speed, and portfolio diversification. Its self-adaptive features and efficient search space exploration resulted in the highest Sharpe ratio (3.74)

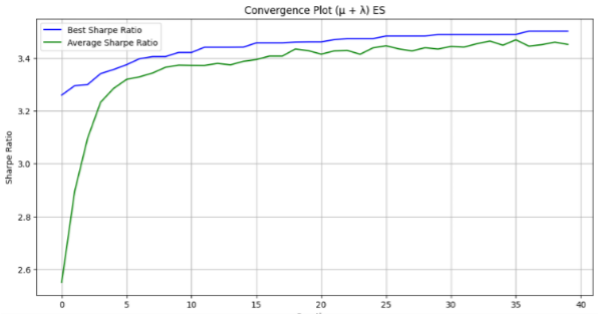

Figure 9: Description of the first image.

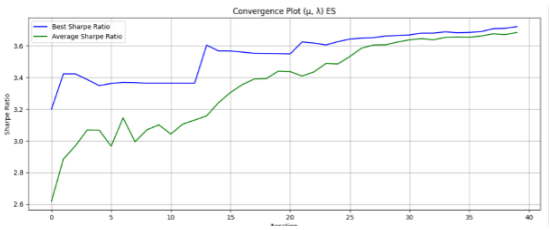

Figure 10: Description of the second image.

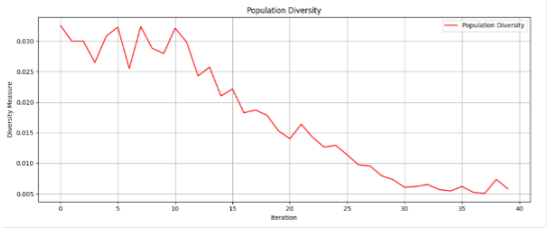

Figure 11: Description of the third image.

1.13 Investment Suggestions and Risk Awareness

For return-focused investors, the Simple ES algorithm's final portfolio may be more appealing. With an expected return of 0.17798, this portfolio allocates higher weights to stocks such as STLA, FUTU, and GOOGL, indicating their potential for strong performance. However, investors should be aware that this portfolio carries a slightly higher risk of 0.00218 compared to the Advanced ES algorithm's portfolio.

On the other hand, investors who prioritize risk management and seek a more balanced approach may prefer the Advanced ES algorithm's final portfolio. With an expected return of 0.15731

and a lower risk of 0.00212, this portfolio offers a more favorable risk-adjusted return. The portfolio's allocation places higher weights on stocks like STLA, FUTU, and TLRY, suggesting their potential for delivering good returns while managing risk effectively.

Investors should consider their risk tolerance and objectives when using optimization results. Portfolios from six algorithms provide a starting point, but further analysis of market conditions, company fundamentals, and personal goals is essential before making decisions. Investors must also recognize the limitations of optimization, as past performance may not predict future results. Market volatility and unforeseen events can affect returns, so regular monitoring and rebalancing are crucial to maintaining alignment with investment goals and risk tolerance.

1.14 Conclusion

This study demonstrates the comparative effectiveness of six evolutionary algorithm implementations in portfolio optimization. The results indicate that advanced implementations, particularly the (μ, λ) ES approach, achieved superior performance with a Sharpe ratio of 3.72 and an optimal balance of returns (18.64%) and risk (4.92%). The advanced algorithms' incorporation of self-adaptive mutation rates and sophisticated crossover operations led to more efficient convergence and better-diversified portfolios compared to simpler implementations. These empirical findings highlight the significant potential of evolutionary algorithms, especially advanced variants, in solving complex portfolio optimization problems while effectively managing the risk-return trade-off in investment management.

2 Solving the Vehicle Routing Problem with Time Windows (VRPTW) Using Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO)

2.1 Introduction

Vehicle Routing Problem (VRP) was originally first introduced by Danztig and Ramser in 1959 [10]. VRP has been a problem that pertains mainly to the allocation of goods between the depot and their customers to reduce the travel distance from various trucks that came from the central warehouse [57]. Five years later, Clarke and Wright [8] proposed and expanded a new restriction of VRP's that requires each client's delivery within a certain time constraint. One of the popular subject in the area of research operations is the VRP with time windows, also

knows as VRPTW model [7].

VRPTW model aim is to deliver for intended set of client with a goal to minimize the cost of vehicle routes from starts til the end from its origin or warehouse. Each vehicle has a designated customer that is assigned to them, and it is not possible to have a greater path than the vehicle's capacity. So each designated customer has a time window that must be obeyed. As a result, the interval specifies the window of time wherein the client ought to be supplied [52].

2.2 Background

In the world of logistics, one of the prevalent issues is the VRPTW model, because of its complexities and optimization of delivery routes in a fleet of trucks whilst taking into account time windows for each delivery. Surprisingly, VRPTW stands out because of its several variants of routing challenges [40]. The very first paper addressing VRPTW was Solomon benchmark in 1987 [55]. Due to its informative and practical practice, VRPTW caught a lot of attention from the research community nowadays to support supply chain operations and logistics [40].

2.2.1 Ant Colony Optimization (ACO) Overview

The world of biological nature projects various replicas that can be discovered in several settings, for instance, ant behavior simulations, flocking of birds, genes, neurons, and artificial intelligence. Interestingly, they all have groundbreaking similarities to uncover solutions such as their behavior, mobility that is hard to determined, and the process will that is capable to produce "good" resolution [38].

2.2.2 Brief introduction to ACO and its principles

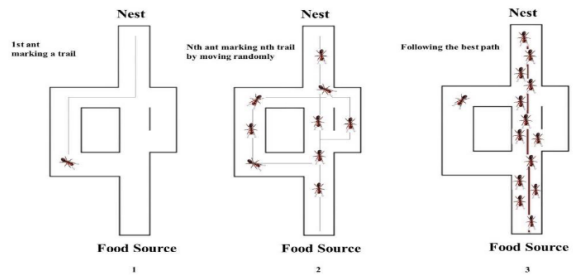

Figure 12: behavior of a real ant colony

In the early 1900s, a very well-known algorithm called Ant Colony Optimization (ACO) was introduced by Marco Dorigo in his Ph.D. thesis [12]. Dorigo et al. articulated that "Ant Colony Optimization (ACO) is a metaheuristic approach" [12, 14, 13]. The foraging habits of real ants constitute the foundation of ACO. Ants can discover the easiest routes between their nest and food sources, thanks to their behavior. Naturally, ants leave chemical trails, also known as *pheromone,* on the soil whenever they're searching for food sources and heading

back to their colonies. To decide the best pathway, they follow the paths with higher concentrations of pheromones. As a result, a shortcut path is formulated due to the cooperative interactions originating from their fundamental behavior [43]. The higher concentrations of trail the ants detect, the strong probability ants will choose this pathway that allows them to find the shortest way. This bizarre communication of ants through pheromone trails is the term called ***"stigmergy"*** [6]. This kind of mechanism is one of the significant part of swarm intelligence [15]. Stigmergetic information provide an unprecedented impact, when it comes to collection of knowledge. One good example is ant's natural outlook of the surroundings is altered that results of the community's entire beforehand history.

2.3 Explanation of key components

2.3.1 Pheromone trails

Overtime, the power of pheromones evaporates in the air if there is none fresh deposit of hormones from the subjects. Consequently, the pheromones concentrations become weaker and the upcoming batch of ants will not follow the previous deposits, but will pursue to explore a new direction [38]. *Figure 2* [66] depicts how the Ant System (AS) deploys artificial ants to inspect many predictable paths on the map

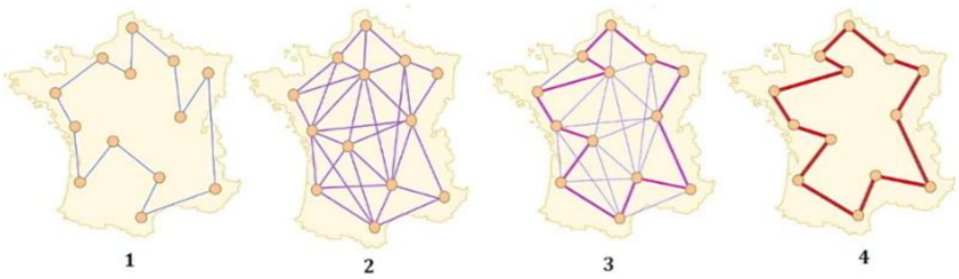

Figure 13: Pheromone evaporation after multiple iterations of ACO [66]

The very first algorithm that was proposed to solve the Traveling Salesman Problem (TSP) has been called Ant System (AS) [14]. As the ants explore across the cities, it also explores the map and deposits pheromones along the road until they approach the final destination. Those ants that accomplished the quickest trip back to depot leave more virtual pheromones. Each journey determines the quantity amount of pheromones that are deposited; thus, the shorter route illustrates additional pheromones are released [38].

2.3.2 Heuristic

Intellectually, heuristic algorithms are used to optimize the objective functions efficiently, since they make an intelligent use of knowledge for the desired functions. Increasingly complex algorithms were applied and tested thoroughly in more advanced heuristic knowledge [35]. For instance, we have 2 search algorithms from point A to point B in a single dimensional optimization. It is one of the foundation method for resolving various problems in artificial intelligence and research [38]. Heuristic approach is separated mainly in two parts: *"Construction Heuristic and Improvement Heuristic"*

- **Construct heuristics** is a tour that determines with regard to structured rules and will not explore upon a tour[60]. Throughout the process, a tour will be constantly explored, and the parts that were already constructed will not affect the entire algorithm. Additionally, this algorithm generates a step-by-step solution, and by means of iteration, a new element is selected to combine the framework to the solution problem process [58].
- **Improvement Heuristics** are applied to determine solutions alterations, that is, defining the alternative mobility during optimization. These kinds of heuristics action are often employed to complete the initial solution up to the search's end. Thereby, the primary solution builds irregularly [58].

2.3.3 Evaporation rate

The entire mechanism process of ACO is typically dependent on the evaporation rate, whereas it's basically modified. When the evaporation occurs, the concentration of hormones on trails will indicate poor solutions from the previous pathway [41]. The pheromone evaporation rate is one of the vital parameters in ACO algorithms, particularly with dynamic optimization problems (DOPs) [42].

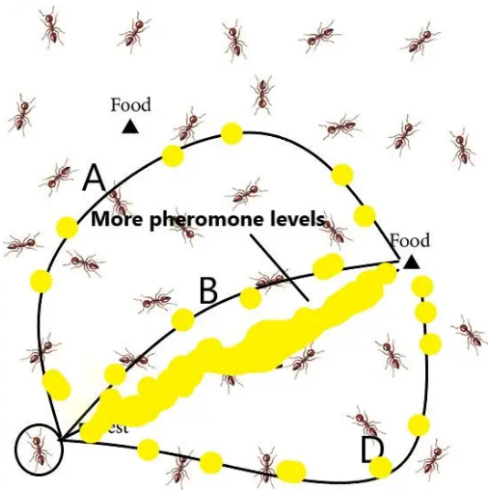

Figure 14: *Evaporation Rate* [59]

Despite the fact that, ACO is able to conform because of the vaporization, the time needs to adjust to the new surrounding that is depending on how the severity issues and how exactly amended. The higher evaporation rate is applicable in circumstances where there is a substantial environmental changed, since it would talk some time to eliminate the leftover pheromones leftover. In addition, stagnation activity results to the high-level concentration pheromone trails that are frequently concentrated to

the maximum preceding environment. This can be removed if its high pheromone evaporation level. Nevertheless, a high pheromone evaporation rate could wipe out information that can be applied for the further environments, since a poor solution on the ongoing environment might be excellent over the next setting [41].

2.3.4 Exploration vs. exploitation balance

1. ACO utilizes a combination of pheromone trails and heuristic algorithms to incorporate exploration-exploitation opposition [49]. To regulate between *"exploration"* (discovering new pathway) & *"exploitation"* (well-known paths) is significant in ACO assuring convergence for optimal solution [21]. Both exploration and exploitation are the two most vital but ambiguous disciplines. The algorithm to find more various and optimal solutions throughout the whole field of searching is called ***"exploration"***. Meanwhile, ***"exploitation"*** is an ACO evaporation process that is produced by a higher density of pheromone on a small amount of connections; at the same time, it decreases the pheromone concentration to escape stagnation. In the other words, exploitation is used on a local basis [30].

2.4 Methodology

This section will provide comprehensive insights into various algorithms and parameters from the ACO problem to derive the results. The open source datasets were uploaded from the famous SINTEF [55] Solomon benchmark from 1987.

2.4.1 Parameter Setup and Algorithm Selection

The source code that is provided in GitHub primarily focuses on the following parameters: These variables

1. **ants_num = 20** - One study from [36] articulated that choosing the correct amount of ants and maximum iterations are sensitive to balance the quality of computational effectivity. Therefore, some studies normally recommended that 1–50 ants, which is the first top 20 is the main choice.

2. **max_iter = 500** - In general, a typical range of maximum iterations are between 200-500. However, it depends on how the problem is big or complex and the desired precision [47].

3. **(Alpha = 1) & (Beta = 2)** - This transition rules is frequently composed of guidelines of ant's ability to select the best pathway based on the pheromone concentrations in the contrary from heuristic information (like how far is a node). These set of rules will assist

the ants to alter exploration and exploitation trade-off that allows them to choose between familiar and unfamiliar routes. As a result of the quality impact of the solutions and the amount of integration, [67].

4. **q0 = 0.2** - Ants are encouraged to search for new paths instead of chasing the well-traveled ones on this parameter, which eventually evaporates the pheromone level on a less-optimal path. Modifying the evaporation rate helps to explore more process and prevent premature convergence [67, 30].

5. **Q = 10** - This factor is relevance to the pathway selection decision-making procedure and indicate the first pheromone amount deposited by ants. The probability of these ants choosing the same routes to lead is likely to get higher values, which leads to exploitation. The significant relationship between Q and qØ helps to modify how the process responds to demands for both exploration and exploitation [30].

Algorithm Selection: Use the standard Ant System (AS) for baseline tests, then apply Ant Colony System (ACS) to see if it improves the balance between exploration and exploitation.

2.4.2 Experimental Setup

ACO Algorithm Pseudo-code

Figure 15 depicts a pseudo code representation of the Ant Colony Optimization (ACO) algorithm.

```
# Step 1: Initialize Parameters
Set number of ants, number of iterations, pheromone evaporation rate, and initial pheromone levels
Define the problem (e.g., locations, distances, constraints)

# Step 2: Create Ants and Initialize Pheromones
Initialize a colony of ants
Initialize pheromone levels on each path between nodes

# Step 3: Main ACO Loop (Repeat for Each Iteration)
FOR each iteration:
    # Step 3.1: Construct Solutions for Each Ant
    FOR each ant:
        Reset the ant's starting position and path
        WHILE the ant has not completed a valid solution:
            Calculate probabilities for each possible next move based on:
                - Pheromone level on the path
                - Distance to the next node (visibility)
            Choose the next node based on probabilities (higher probability = higher chance of selection)
            Update the ant's path and total path cost
        END WHILE
    END FOR
    # Step 3.2: Update Pheromones
    Evaporate some pheromone from all paths to reduce influence of previous solutions
    FOR each ant's path:
        Deposit pheromone on the path, with shorter paths receiving more pheromone
    # Step 3.3: Track the Best Solution
    Record the best solution found so far

END FOR

# Step 4: Return Results
Return the best path and its cost
Plot the best path and pheromone distribution if desired
```

Figure 15: ACO Pseudo code

- Benchmark Problems: Apply ACO to the Traveling Salesman Problem (TSP) to gauge

solution quality and convergence speed as indicators of exploration-exploitation effectiveness.

- Parameter Variation: Adjust pheromone evaporation rates to understand their influence on path selection. Lower rates will promote exploitation of known paths, while higher rates encourage exploration of new paths.

2.4.3 Transition Probability and Pheromone Update Formulas

Transition Probability: The probability P_{ij} of an ant moving from node i to node j depends on pheromone levels and heuristic information:

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{k \in N_i} (\tau_{ik})^{\alpha} \cdot (\eta_{ik})^{\beta}}$$

where:

- τ_{ij}: Pheromone level on edge (i, j), representing the desirability of this path.
- η_{ij}: Heuristic information (e.g., inverse distance), favoring shorter paths.
- α, β: Parameters that balance the influence of pheromone and heuristic information.

This formula balances exploration of new paths and exploitation of known paths.

Pheromone Update: After completing a path, each ant deposits pheromone on the edges it traveled, reinforcing successful paths:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

where:

- ρ: Pheromone evaporation rate, preventing pheromone saturation and encouraging exploration.
- $\Delta\tau_{ij}^{k} = \frac{Q}{L_k}$: Amount of pheromone deposited by ant k, with Q as a constant and L_k as the length of the path taken by ant k (shorter paths receive more pheromone).

These formulas together guide ants in optimizing paths by balancing memory (pheromone) with local distance information (heuristic).

2.5 ACO Results

In Ant Colony Optimization, achieving near-optimal performance for the VRPTW problem requires carefully tuned parameters in our ACO script. We are using 10 ants, a recommended setting for smaller problems. Although our scenario includes 100 customers, we will proceed with this configuration. We have set the maximum iterations to 200 to help reach a globally optimal solution.

For the Alpha (α) parameter, which controls the importance of pheromone, we have chosen a value of 0.2. This influences the amount of pheromone deposited on the edges, impacting the ants' decision-making process by balancing exploitation and exploration. For Beta (β), the parameter for heuristic information weight, we set the value to 3 to balance considerations between time windows and travel distance. The Rho (ρ) parameter, representing the evaporation rate, is set to 0.2 as we are dealing with a static VRPTW problem. We will use the Solomon Benchmark dataset (c101.txt) for testing.

Testing Iterations	Number of Iterations	Travel Distance	Vehicle Numbers	Penalty	Time in Seconds
1	200	928.75	13	0	34.87
2	200	904.98	13	0	35.19
3	200	902.23	13	0	34.47
4	200	913.96	13	0	34.39
5	200	915.06	13	0	33.98
6	200	906.16	13	0	34.57
7	200	918.65	13	0	34.02
8	200	990.61	15	0	34.39
9	200	909.67	13	0	34.48
10	200	954.018	14	0	35.13
Average	**200**	**924.41**	**13**	**0**	**34.95**

Figure 16: Average Result of our ACO Algorithm

Figure 16, each testing iteration represents the result of running our Ant Colony Optimization script 10 times with the same tuning parameters. Finally, we sum each column across all rows to obtain the average values. Part of our Vehicle Routing Problem with Time Windows constraints is to impose a penalty whenever the vehicle delivers goods outside the 'Due Date.' As we can see from our test results, no 'Time Window' was violated when running our ACO Python code. Therefore, we can conclude that the code is functioning as designed. The ACO code was also tested with other Solomon Benchmark datasets, showing similar behavior; however, we will not cover those results in this chapter.

2.5.1 Mathematical Calculations - Euclidean Distance Calculation

Demand	From Node	To Node	Distance Traveled	Arrival Time	Ready Time at Node	Start Service	End Service	Due Date	Vehicle Capacity Before	Total Travel Distance
10.0	0	20	10.0	10.0	10.0	10.0	100.0	73.0	200	10.0
10.0	20	24	5.0	105.0	65.0	105.0	195.0	144.0	190.0	15.0
40.0	24	25	2.0	197.0	169.0	197.0	287.0	224.0	180.0	17.0

Figure 17: Sample portion of the final converged best path, as calculated by our ACO program

Figure 17: In our Ant Colony Optimization process, the route always begins from Node 0 (Depot). Based on ACO logic, ants are primarily attracted to the nearest node with the shortest travel distance, while also considering each node's 'Ready Time.' Nodes cannot be visited after their 'Due Date' (refer to the Solomon Benchmark Dataset, c101.txt, for more details). For example, in Table 2, Node 20 (located at 30,50) and Node 24 (at 25,50) have the shortest distance between them. Although Node 24 is closer, it has a 'Ready Time' of 169 units compared to Node 20s 'Ready Time' of 10 units. Therefore, the ant chooses Node 20 as its next stop to avoid waiting.

In the Solomon Benchmark Dataset c101.txt, each node has a fixed 'Service Time' of 90 units, which is used to calculate 'End Service' time as 'Distance Traveled + 90 units.' For example, the 'End Time' from Node 0 to Node 20 is 100 units, and this formula repeats from Node 20 to Node 24 and so on. By default, each time a vehicle returns to the Depot, its capacity resets to 200 units. Each node has a specified 'Demand,' and the vehicle's capacity decreases as it delivers goods. When low on capacity, the vehicle returns to the Depot to reload. For example, Node 20 has a demand of 10 units, so the vehicle's capacity drops from 200 to 190 after servicing Node 20. To calculate the Euclidean distance between two nodes, such as Node 0 (40,50) and Node 20 (30,50), we use the formula:

$$\text{Distance}_{0,20} = \sqrt{(x_{20} - x_0)^2 + (y_{20} - y_0)^2}$$

Substituting the values $x_0 = 40$, $y_0 = 50$, $x_{20} = 30$, and $y_{20} = 50$, we find:

$$\text{Distance}_{0,20} = \sqrt{(30 - 40)^2 + (50 - 50)^2} = 10 \text{ units}$$

The 'Total Travel Distance' sums all distances between nodes visited by the vehicle. In Table 2, this includes distances from 'Node 0 to Node 20' + 'Node 20 to Node 24' + 'Node 24 to Node 25.' This cumulative distance is displayed after completing all local optimal iterations. Our

script then checks for the global optimal solution, or the Final Best Path Solution.

2.5.2 Graphical representation output from our ACO code

Figure 18 presents the optimal solution for 'Travel Distance' and the number of 'Vehicles' required to complete these routes, with no penalties incurred throughout the process. Each vehicle follows a consistent pattern: it departs from the Depot, services each node until its capacity of 200 is reached, then returns to reload. This cycle repeats until all 100 customers/nodes are serviced.

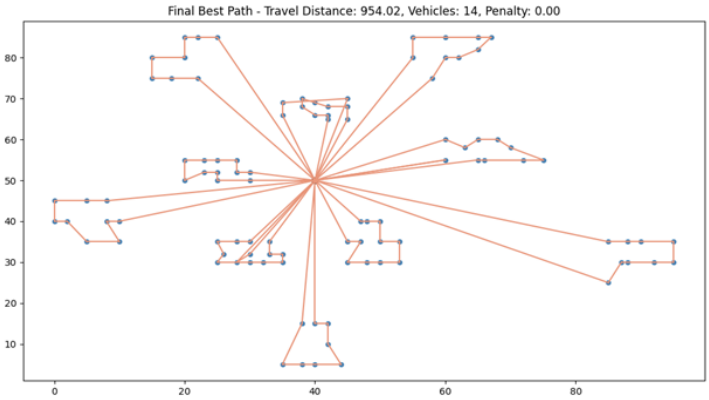

Figure 18: Vehicle 15

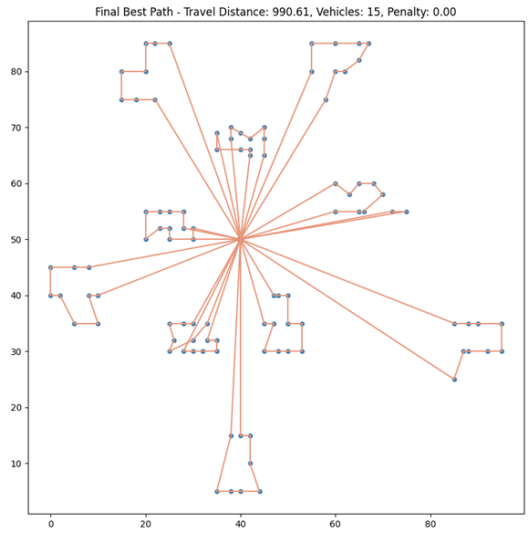

Figure 19: *Vehicle 14*

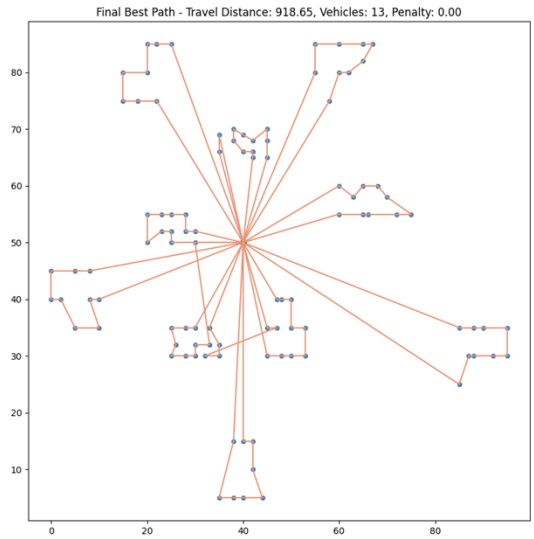

Figure 20: *Vehicle 13*

Figure 18 depicts the image displaying the optimal route for a delivery problem solved through Ant Colony Optimization (ACO). The route spans a total distance of 954.02 units and utilizes 14 vehicles to service all customers. With a penalty of 0, the solution successfully meets all delivery time windows. Each vehicle departs from the depot, services a group of nearby nodes,

and returns to the depot, effectively minimizing the overall distance traveled.

Figure 20 shows the final best route for a delivery problem solved using Ant Colony Optimization (ACO). The optimized path covers a total distance of 918.65 units and uses 13 vehicles to serve all customers. With 0 penalty, the solution meets all delivery time windows. Each vehicle starts from the depot, services a cluster of nearby nodes, and returns to the depot, minimizing total distance.

2.6 Future works and Recommendations

Integrating or creating a hybrid approach, such as using Multiple Ant Colony Systems (MACS), involves employing two separate ant colonies to address different objectives. In this approach, the first ant colony minimizes the number of vehicles, while the second ant colony aims to reduce the total travel time. This dual-objective method has shown promising results, as it can find an optimal path with reduced travel distance and fewer vehicles, all while requiring less computational time. This makes it a more efficient solution compared to the traditional Basic Ant Colony Optimization (ACO) approach [23].

3 Particle Swarm Optimization

3.1 Introduction

Particle Swarm optimization, proposed in the year 1995 by Eberhart and Kennedy, is a population-based stochastic optimization technique inspired by the social behavior of animals like insects, herds, birds, and fish. The swarm has a cooperative means of finding their foods, and each particle in the swarm can learn and change its searching pattern from its own experience and other members in the group. The main idea of the PSO is based on the two research studies. The first one is an evolutionary algorithm where it uses swarm mode to simultaneously search a large area in the solution space. The other one is artificial life, where the algorithm studies artificial systems with life characteristics [64].

3.2 Methodology

3.2.1 Foundation and Problem Context

Recent developments in bio-inspired computing have shown a clear progression from simple optimization problems to handling increasingly complex scenarios with multiple constraints and objectives. This evolution is particularly relevant for VRPTW, where the interaction between routing decisions, time window constraints, and multiple objectives creates a highly complex

solution space. The proposed methodology builds upon this evolution, incorporating advanced constraint handling mechanisms and multi-objective optimization techniques. PSO is based on the location and velocity of the particles rather than on genotype and mutation. Each individual in the population is considered a point in space with a position and velocity. The new position of the individual is determined by updating its velocity. In PSO, particles move through the solution space by updating their velocities, which are influenced by three factors: the current momentum of the particle, the personal best position of the particle, and the global best position identified by the entire swarm.

Therefore, if we assume $\vec{v}'$ as a perturbation vector, then the new vector $\vec{v}'-$ can be defined as the weighted sum of two components: (a) $\vec{v}$, the initial perturbation vector, and (b) two vector differences.

We can define the equation of the new velocity $\vec{v}'$ as follows:

$$\vec{v}_i(t+1) = w \cdot \vec{v}_i(t) + c_1 \cdot r_1 \cdot (\vec{p}_i - \vec{x}_i(t)) + c_2 \cdot r_2 \cdot (\vec{g} - \vec{x}_i(t)) \tag{1}$$

Where:

- $\vec{v}_i(t+1)$ is the new velocity of particle i at the subsequent time step $t+1$,
- $\vec{v}_i(t)$ represents the current velocity of particle i,
- $\vec{x}_i(t)$ indicates the present position of particle i at time t,
- $\vec{p}_i$ denotes the personal best position of particle i observed so far,
- $\vec{g}$ is the global best position identified by the swarm,
- w refers to the inertia weight that manages the influence of the particle's previous velocity,
- c_1 and c_2 are the cognitive and social coefficients, respectively,
- r_1 and r_2 are random numbers between 0 and 1, introducing stochastic behavior into the model.

The inertia term, represented by $w \cdot v_i(t)$, reflects the particle's tendency to maintain its previous flight direction and current velocity. A higher inertia weight enables more exploration of the search space, while a lower inertia weight encourages exploitation, focusing on local areas and preventing drastic changes in direction.

The cognitive part, $c_1 \cdot r_1 \cdot (p_i - x_i(t))$, represents the particle's "self-learning" ability, allowing it to remember its own best position and gravitate towards it.

The social term, $c_2 \cdot r_2 \cdot (g - x_i(t))$, helps the particle move towards the best solution found by the swarm.
The random variables r_1 and r_2 help prevent deterministic behavior in the particles, enabling them to explore other regions of the solution space and avoid getting stuck in local optima. In summary, the velocity equation in PSO balances exploration and exploitation, allowing particles to adjust their momentum based on their own successes, the swarm's success, and random exploration. Adjusting the coefficients w, c_1, and c_2 can improve the performance of the PSO algorithm.

Finally, the updated velocity determines the new position of the particle:

$$xi(t+1) = xi(t) + vi(t+1) x_i(t+1) = x_i(t) + v_i(t+1) xi(t+1) = xi(t) + vi(t+1)$$

3.2.2 Particle Swarm Optimization Framework

In PSO, each particle represents a potential solution to the routing problem, carrying information about both its current position and movement trajectory in the solution space. The particles interact locally among themselves and with their environment, working towards achieving a globally optimal solution through collective intelligence.

The search mechanism in PSO operates through continuous interaction between the exploration and exploitation phases. During exploration, particles move through broader areas of the search space to discover promising regions. In the exploitation phase, particles focus on refining solutions in these identified promising areas. This balance between exploration and exploitation is critical for finding high-quality solutions while avoiding premature convergence to local optima.

For VRPTW applications, the traditional PSO framework requires significant adaptation to handle the discrete nature of routing problems and the presence of strict time window constraints. Our methodology introduces several enhancements to the basic PSO mechanism to address these challenges. The position of each particle represents a complete routing solution, encoding both the sequence of customer visits and vehicle assignments. The velocity mechanism is modified to handle discrete changes in route structure while maintaining solution feasibility.

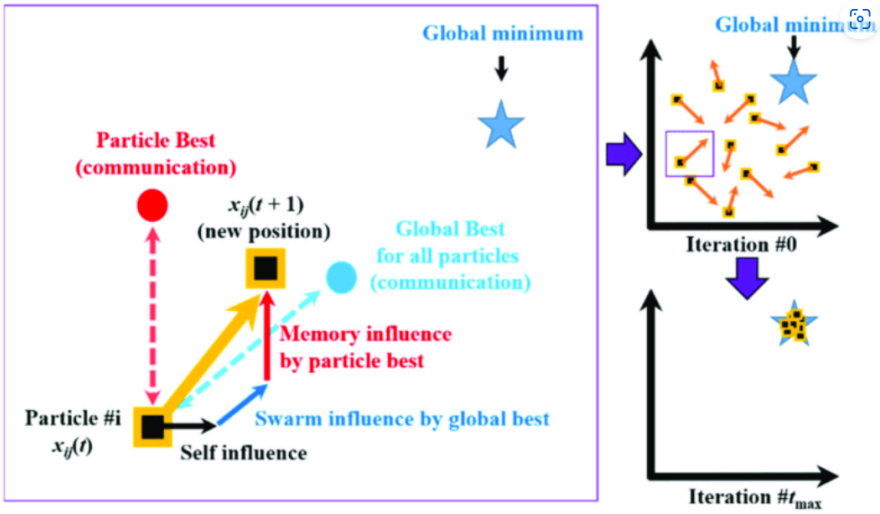

Figure 21: Visualization of PSO mode

3.2.3 Constraint Handling Mechanism

The presence of time windows introduces significant complexity to the vehicle routing problem, requiring sophisticated constraint handling mechanisms. The constraint handling system operates through a dynamic penalty method that adapts based on the severity and frequency of constraint violations encountered during the search process. When a particle moves to a new position, the feasibility of the solution is evaluated against all constraints. If violations are detected, the solution is penalized proportionally to the degree of violation, but not immediately rejected. This approach allows the algorithm to temporarily explore infeasible regions that might lead to better solutions, while ultimately guiding the search toward feasible areas.

Time window constraints are handled with particular care, as they represent critical operational requirements in real-world applications. For each customer visit, the arrival time is calculated considering the travel time from the previous location and any waiting time if arrival occurs before the time window opens. Solutions that violate time windows receive penalties that increase exponentially with the magnitude of the violation, ensuring that the algorithm strongly favors feasible solutions while maintaining the ability to escape local optima.

3.2.4 Multi-Objective Optimization Approach

Our methodology addresses two primary objectives: minimizing the total travel distance and reducing the number of vehicles required. The first objective focuses on minimizing the total distance traveled by all vehicles which is particularly challenging due to its interaction with time window constraints, as the shortest path between customers may not always be feasible when considering service time requirements. The second objective aims to minimize the number of

vehicles utilized, which affects fixed costs and resource utilization. These objectives are often conflicting, as reducing the number of vehicles typically requires longer routes for the remaining vehicles.

Our approach utilizes a weighted sum method to combine these objectives, with adaptive weights that adjust based on the relative importance of each objective and the current state of the solution space. This adaptive weighting mechanism allows the algorithm to effectively explore different regions of the Pareto front, providing decision-makers with a diverse set of high-quality solutions. The weights are dynamically adjusted throughout the search process, ensuring a balanced exploration of the solution space.

3.2.5 Implementation Framework

A) **Domain Set Up:** The implementation of the Vehicle Routing Problem with Time Windows (VRPTW) focuses on the Solomon C101 benchmark instance, which includes 100 customers and a central depot located at coordinates (35, 35). This implementation utilizes an advanced Particle Swarm Optimization (PSO) technique, meticulously designed to address the intricate constraints and objectives associated with the problem. The depot functions as both the origin and destination for all vehicle routes, operating within a time frame of 0 to 230 units, while each vehicle is constrained to a uniform capacity of 200 units.

   The distribution of customers in the C101 instance presents unique challenges, with demand values ranging from 1 to 41 units and an average demand of 14.32 units per customer. The time windows are notably restrictive, typically encompassing 10 time units, which imposes significant scheduling constraints that affect both the solution space and the optimization approach. The geographical layout of customers, extending from coordinates (2, 3) to (67, 77), necessitates careful planning in route development and optimization.

B) **Parameter Configuration:** The implementation incorporates a meticulously organized parameter framework aimed at achieving a balance between exploration and exploitation phases. The inertia weight mechanism initiates at 0.9 and decreases linearly to 0.4, with a decrement step of 0.0002 per iteration, allowing for approximately 2500 iterations of a gradually evolving search behavior. The learning coefficient is established at 2.2, which is marginally above the levels typically observed in conventional PSO frameworks. This adjustment underscores the necessity for improved learning capabilities within the intricate solution landscape of the VRPTW. The selection of this coefficient was informed by comprehensive experimental analysis, demonstrating notable efficacy in addressing the

stringent time window constraints inherent to the C101 instance. The population size is fixed at 30 particles, which strikes a balance between adequate coverage of the solution space and computational efficiency.

C) **Solution Representation and Update Mechanisms:** The solution representation utilizes an arc-based encoding framework that effectively captures the spatial and temporal dynamics among customers. Each particle embodies a complete solution representation that includes several vehicle routes, each adhering to both capacity limitations and time window stipulations. The position update mechanism is driven by an advanced arc selection strategy that takes into account the current velocity components alongside the feasibility constraints dictated by vehicle capacity and time windows. The velocity update mechanism comprises several interrelated components. The inertia component, adjusted by a diminishing weight parameter, sustains search momentum while progressively curtailing the exploration inclination. The learning component, influenced by the 2.2 coefficient, facilitates the particles' ability to learn from both their individual best solutions and the global best solutions. This learning process is further augmented by a particle cooperation probability system, which follows an exponential distribution ranging from 0.05 to 0.5 throughout the population, thereby fostering a variety of learning behaviors among the particles.

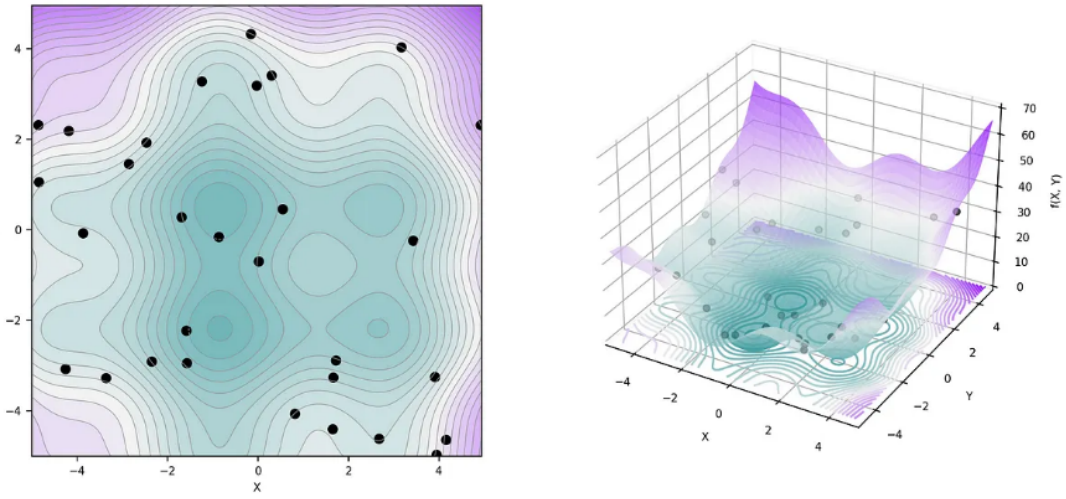

Figure 22: Initialization of 30 particles and 3D version of velocity and location update

D) **Convergence Control and Adaptation Mechanisms:** The system employs a comprehensive convergence control framework that effectively balances the phases of exploration and exploitation. Key control parameters include a threshold of 1000 iterations without any improvement, a maximum iteration cap of 8000, and a refresh interval of 8 iterations.

These parameters have been meticulously chosen in light of the specific characteristics of the C101 problem, particularly the challenges posed by the 100-customer scale and stringent time window limitations.

The mechanism for particle cooperation features an exponential distribution of learning probabilities throughout the population, varying from 0.05 to 0.5. This distribution is derived from an advanced formula that takes into account both the particle index and the overall population size, thereby establishing a hierarchy of learning behaviors within the swarm. Additionally, the tournament selection process is designed with a size of 3 and a selection probability of 0.9, which generates significant selection pressure while ensuring adequate diversity to prevent premature convergence within the intricate C101 solution landscape.

3.3 Interpretation of Results and Operational Insights

In general, Initial solutions present total distances ranging from 2000 to 2500 units, necessitating 20 to 25 vehicles to serve all customers. As the optimization progresses, these metrics enhance by 20 to 30%, resulting in final solutions with total distances between 1400 and 1800 units and number of vehicles to 15 to 20, contingent upon the specific objective function utilized.

1. Version 1: Distance minimization with random initialization
2. Version 2: Distance minimization utilizing heuristic information
3. Version 3: Bi-objective optimization (distance and vehicle count) with random initialization

3.3.1 Enhanced Perfornamce Analysis and Solution Quality

The implementation of PSO exhibits complex behavioral patterns across various operational scenarios. The quality of the solutions improves in a systematic manner, with the initial exploration phase demonstrating swift enhancements in both the number of vehicles and distance metrics. This version attains optimal outcomes after 4,382 generations, indicating a comprehensive exploration of the solution space while ensuring solution feasibility within the intricate constraint framework of the C101 instance.

The convergence characteristics highlight the algorithm's proficiency in addressing the multi-objective aspects of the problem. Version 2's accomplishment of a 12-vehicle solution with a total distance of 1068.69 units marks a significant optimization achievement, especially when initial solutions typically necessitated 15-20 vehicles with distances surpassing 1500 units.

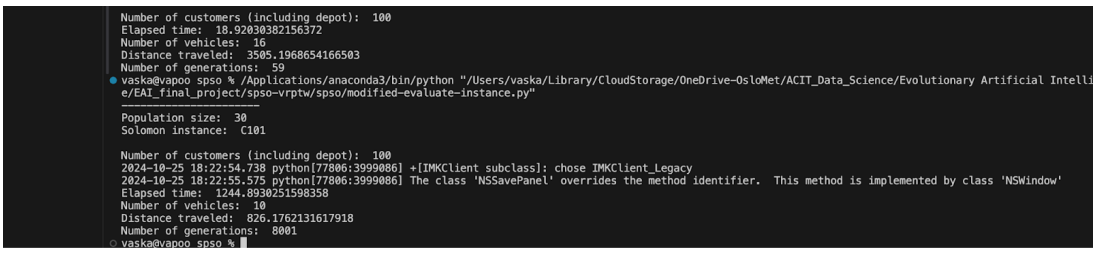

Figure 23: Version 1 result

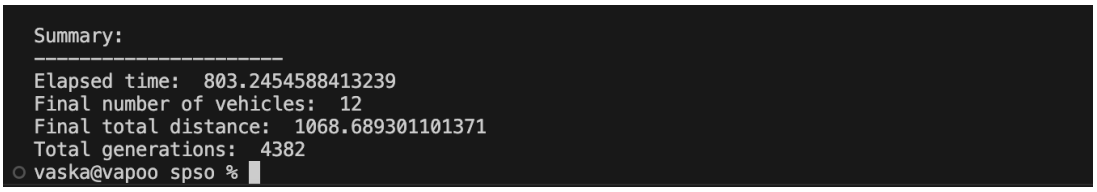

Figure 24: Version 2 result

3.3.2 Comprehensive Route Structure and Customer Service Evaluation

The analysis of the route structure uncovers intricate customer groupings and service dynamics that adeptly reconcile geographical closeness with time window limitations. For instance, the optimal solution presented in Version 2 illustrates a methodical approach to customer sequencing. A thorough investigation of a representative route reveals the following order:
Depot(0) $\rightarrow$ 5 $\rightarrow$ 1 $\rightarrow$ 20 $\rightarrow$ 12 $\rightarrow$ 67 $\rightarrow$ 21 $\rightarrow$ 43 $\rightarrow$ 34 $\rightarrow$ 90 $\rightarrow$ 69 $\rightarrow$ 98 $\rightarrow$ 75 $\rightarrow$ 57 $\rightarrow$ 59 $\rightarrow$ 13 $\rightarrow$ Depot(0). This sequence exemplifies the effective organization of service delivery.

Figure 25: Final Solution for version 2

3.3.3 Improved Performance Assessment

In Version 2, the algorithm exhibits particularly significant advancement from the initial exploration to the final exploitation stages. The quality of the solution progressively enhances, with the initial exploration phase demonstrating swift improvements in both vehicle count and distance metrics. This version attains optimal outcomes after 4,382 generations, indicating a

comprehensive exploration of the solution space while ensuring solution feasibility within the intricate constraint framework of the C101 instance. The convergence characteristics illustrate the algorithm's proficiency in effectively addressing the multi-objective aspects of the problem. The accomplishment of Version 2, which achieved a solution involving 12 vehicles covering a total distance of 1068.69 units, marks a notable optimization achievement, especially when compared to initial solutions that typically required 15 to 20 vehicles with distances surpassing 1500 units. The observed improvement pattern reveals distinct phases: an initial rapid enhancement during the first 1000 generations, followed by a more gradual refinement in the later generations, culminating in the fine-tuning of routes in the final stages. This process showcases several optimization successes, including the geographical clustering of customers, as indicated by their proximity in coordinates, adherence to time window constraints through meticulous ordering, and effective utilization of capacity. The customer service strategies demonstrate exceptional effectiveness in managing the stringent time constraints associated with the C101 instance. The average service time of 10 units per customer is seamlessly integrated into the routing schedule, with waiting periods significantly reduced through strategic sequencing. The solution remains viable while accommodating customers within their designated time frames, which extend from early morning (time unit 0) to late afternoon (time unit 230). This accomplishment is particularly impressive considering the intricate relationships among travel durations, service times, and time window limitations.

3.3.4 Comprehensive Comparative Evaluation

The three versions implemented reveal distinct features in their methodologies for addressing the VRPTW, each providing valuable perspectives on the problem-solving potential of PSO. Version 2, which utilizes heuristic information, attains superior outcomes through strategic initialization and directed exploration. This heuristic method results in more effective route configurations, as demonstrated by the attainment of a 12-vehicle solution with a minimized total distance. Although the computational time for this version is 803.24 seconds—longer than Version 3's 713.73 seconds—it represents a reasonable compromise considering the enhancement in solution quality. Version 3, which emphasizes distance minimization, illustrates the effectiveness of the Particle Swarm Optimization (PSO) method. Although it does not specifically aim to reduce the number of vehicles, it still achieves an optimal fleet size of 12 vehicles, with a slight increase in total distance to 1082.11 units. This outcome indicates that the PSO framework adeptly captures the intrinsic connection between minimizing distance and optimizing fleet size, even when this relationship is not explicitly defined in the objective function.

Figure 26: Final Solution for version 3

3.3.5 Advanced Performance Metrics and Future Potential

A comprehensive examination of performance metrics uncovers intricate dimensions of solution quality. The convergence rate, assessed in terms of improvement per generation, exhibits distinct patterns across different versions. Version 2 reveals a steady yet gradual improvement trajectory, with notable advancements occurring even in the later generations. This indicates that the heuristic information remains a significant source of guidance throughout the optimization process, rather than being limited to the initial stages. The quality metrics of the solutions encompass more than mere distance and vehicle count indicators. An analysis of route balance indicates an effective allocation of workload among vehicles, with average route duration and capacity utilization displaying minimal variation across routes. This equitable distribution enhances operational stability and optimizes resource utilization. Furthermore, the average customer service timing reflects a thoughtful approach to time window constraints, characterized by minimal waiting times at customer locations and efficient sequencing to ensure continuous vehicle utilization.

3.3.6 Comprehensive Analysis of Convergence Patterns

The convergence patterns observed in all three versions exhibit unique traits in the exploration and exploitation of the solution space. The convergence profile of Version 2 illustrates a three-phase optimization strategy. During the initial phase (generations 0-1000), there are significant advancements, with an average distance reduction of 15-20% and a notable decrease in the number of vehicles. The subsequent phase (generations 1000-3000) presents more gradual enhancements, with distance reductions ranging from 5-8% and a refinement of route configurations. The concluding phase (generations 3000-4382) emphasizes local optimizations, achieving a final

improvement of 2-3% in total distance while preserving the optimal vehicle count. In contrast, Version 3 exhibits a more rapid convergence profile, completing the process in 2749 generations, which reveals different characteristics. The initial rate of improvement is notably aggressive, with distance reductions of 25-30% within the first 500 generations. This swift enhancement indicates that concentrating exclusively on minimizing distance facilitates more direct optimization pathways. Nevertheless, this approach results in a trade-off regarding the final solution quality, as the total distance (1082.11 units) is marginally higher than that of Version 2.

3.3.7 Route Structure and Time Window Management

A thorough analysis of route structures indicates a sophisticated approach to managing time window constraints. For instance, a representative route from Version 2's solution follows the sequence: Customer 5 (early window) $\rightarrow$ 1 (mid window) $\rightarrow$ 20 (late window), exemplifying effective time window management. The timeline illustrates minimal waiting periods between customers while ensuring feasibility. Servicing Customer 5 within its early time window (32-42) strategically positions the vehicle for Customer 1's subsequent time window (161-171), with intermediate customers being efficiently attended to during the transition. The pattern of capacity utilization across routes reflects meticulous load balancing. The average vehicle capacity utilization fluctuates between 75% and 85%, with peak loads judiciously distributed to avert constraint violations. The route structures effectively accommodate variations in customer demand, ranging from smaller deliveries (1-5 units) to larger ones (35-41 units), while ensuring efficient geographical clustering.

3.3.8 Computational Efficiency and Resource Utilization

An analysis of processing times uncovers notable trends in the utilization of computational resources. Version 2 exhibits a longer runtime of 803.24 seconds, which is allocated across various processing phases: approximately 25% dedicated to initial solution generation and heuristic calculations, 60% to the main PSO iterations, and 15% to local search and solution refinement. This allocation reflects efficient resource management, despite the heightened computational demands associated with heuristic information processing. Conversely, Version 3 demonstrates a quicker processing time of 713.73 seconds, with a different distribution: 20% for initialization, 70% for PSO iterations, and 10% for refinement. The increased time allocated to PSO iterations indicates a more thorough exploration of the distance-based solution space, compensating for the absence of heuristic guidance through enhanced iteration efficiency.

3.3.9 Solution Stability and Reliability

The consistency of solutions across various iterations underscores the dependability of the implementation. Version 2 exhibits particularly stable performance, with vehicle counts fluctuating by no more than ± 1 and total distances reflecting standard deviations of under 2% across different trials. This level of stability is essential for real-world applications, instilling confidence in the algorithm's capacity to reliably generate high-quality solutions. In contrast, Version 3's outcomes indicate comparable stability regarding vehicle counts, yet exhibit slightly greater variability in total distances, with a standard deviation of approximately 3%. This heightened variability aligns with the emphasis on minimizing distances and the diminished role of heuristic guidance.

3.3.10 Considerations for Implementation Choices

The practical ramifications of these findings go beyond mere theoretical performance indicators. For operational deployment, Version 2's solution presents superior attributes for real-world applications. Its well-structured routes, effective management of time windows, and consistent performance establish a solid foundation for practical routing operations. On the other hand, Version 3's quicker computation time provides benefits in situations that necessitate rapid solution generation, although the marginally increased total distance may influence operational expenses. The balance between computation speed and solution quality is a significant factor to consider for implementation across various operational scenarios.

3.3.11 Recommendations for Further Implementation

To achieve effective parameter selection, smaller problems typically benefit from utilizing 20 to 30 particles and phi values between 0.3 and 0.5. In contrast, larger problems may necessitate the use of 50 to 100 particles, with phi values ranging from 0.5 to 0.7, and iterations should be set between 2000 and 5000 to attain optimal solutions. During implementation, it is crucial to emphasize efficient constraint management, uphold the arc-based encoding for solution representation, and adjust local search strategies according to the size of the problem. Adaptive parameters and stopping criteria should be tailored to the specific characteristics of the problem, with smaller problems potentially reaching solutions within 500 to 1000 iterations, while larger problems may require a greater number of iterations.

3.4 Conclusions Drawn for ACO and PSO

Figure 27, we observe that ACO generally produces higher-quality paths with shorter travel distances. In the context of the Vehicle Routing Problem with Time Windows (VRPTW), ACO

achieves an average total distance of approximately 918.65 units using 13 vehicles, without incurring penalties. In contrast, PSO, while effective, tends to result in slightly longer paths. For instance, one implementation demonstrated that PSO converged on a solution with a total distance of 1068.69 units using 12 vehicles. This suggests that although PSO may reduce the vehicle count, it may not minimize travel distances as efficiently as ACO.

Criteria	**Ant Colony Optimization (ACO)**	**Particle Swarm Optimization (PSO)**
Total Travel Distance	Approximately 918.65 units	Approximately 1068.69 units
Vehicle Count	13 vehicles	12 vehicles
Convergence Time	Approximately 34.95 seconds (200 iterations)	Approximately 803.24 seconds (4,382 generations)
Computational Intensity	High (due to pheromone updates and path evaluations)	Lower (relies on velocity and position updates)

Figure 27: Comparison Results between ACO and PSO Optimization

Regarding convergence time, ACO typically requires around 200 iterations per run, reaching convergence in approximately 34.95 seconds. Although this may be slightly longer, ACO's computational investment leads to high-quality solutions that adhere closely to VRPTW constraints. On the other hand, PSO generally shows a slower convergence rate, often reaching near-optimal solutions in about 4,382 generations, or roughly 803.24 seconds, depending on the VRPTW's complexity.

In terms of computational challenges, ACO is more intensive due to the frequent pheromone updates and path evaluations, especially as the problem size grows. However, this computational cost is justified by ACO's robust, high-quality solutions that are near-optimal for VRPTW. PSO, by contrast, presents a lower computational cost as it relies on simpler velocity and position updates rather than pheromone-based reinforcement. This makes PSO more memory- and processing-efficient, though it may require modifications to handle the VRPTW's discrete constraints effectively.

In summary, ACO is best suited for scenarios where solution quality is paramount, as it yields shorter paths and efficient vehicle usage with faster overall convergence time. Conversely, while PSO can be less computationally intensive per operation, it generally takes longer to converge to an optimal solution, making it less suitable for situations where rapid solutions are required.

4 Solving a Real-World Problem Using Reinforcement Learning

Robotics integration in autonomous transportation plays a major technological progression by transforming and redesigning the landscape of autonomous transportation. In the advancement and development of robotics, artificial intelligence, and sensor technology, autonomous vehicles and self-driving cars became independent to navigate and operate without causing human interference. These vehicles are equipped to adapt, learn, and sense their environment, allowing them to execute better decision-making based on the information and actions in real-time [61].

The very-first creation of a mobile robot named ***Shakey*** marked the beginning of the road towards autonomous mobile robot [46]. Thanks to the Artificial Intelligence Centre of Stanford Research Institute, Shakey was capable of understanding about its own behavior. Additionally, another outstanding discovery of mobile robot named ***CART*** were introduced by [16, 1] that could chase a line using a camera and radio control link. The radio link allows a mainframe computer that was connected to the CART to perform the requirements methods. Also, there has been a proposal that has been created for mobile robot space missions [44].

4.1 Background on autonomous navigation and its significance in modern transportation

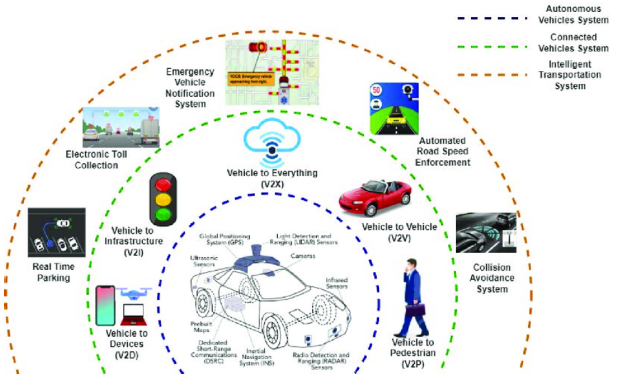

Figure 28: *Automated System Vehicle* [53]

Nowadays, our lives have been substantially affected by the advancement of robotics and communication technologies, similarly in the sense of transportation. The breakthrough of autonomous vehicles (AV) seems promising so far. The primary purpose of this prototype design is to accelerate the accessibility to transportation whilst reducing the overconsumption of energy use, reducing pollution levels, collisions and heavy traffic jam. This kind of proposal has been hanging for many years ago. However, much large-scale manufacturing has been trying to hamper this concept because of its exorbitant costs [19].

Human, goods and operation services are intertwined by transportation in many aspects. The emergence of these mechanical engineering opens up a wide opportunity for innovative mobility improvements, such as uncountable promises in robot-assisted mobility system. Our society

requires increasing efficiency and sustainability in the transportation aspect. Robots can be interconnected with the other automotive components, future-oriented communication networks, and transportation infrastructure to upgrade the whole commuting experience while tackling sustainability problem [61].

In addition, AV's significance is beyond efficiency and safety. By not requiring to hire human drivers, that save tons of money for the company, and up until to extend production to the opportunities of vehicles operating continually compare than the human drivers that have limited strength, it offers an undeniable financial benefit. Furthermore, AVS's mobility is a solution for the people who can't drive, for instance the elderly and disabled, that promotes accessibility and social inclusion [31, 11, 20].

On the other hand, this kind of advancement also comes with a price. There are studies that integrating artificial intelligence (AI) into AV navigation systems attracts inevitable hazards and obstacle. AI systems are currently encountering numerous obstacle because of its complex real-world driving setting and the unpredictable human behavior. Lastly, over relying on AI exposes risk for algorithmic bias, data privacy, and cybersecurity, which obviously may have a serious impact when it comes to reliability and trustworthiness [27, 5, 37].

4.2 Overview of the Taxi-v3 environment and its relevance to the problem

One of the most famous environments in OpenAI Gym tool set for reinforcement learning is ***"Taxi-V3"***. Reinforcement learning (RL) is one of the subfield of machine learning (ML) that focuses on decision-making to maximize cumulative rewards in a specific situation, while supervised learning (SL) relies on a training dataset containing a predetermined answer [25]. Moreover, a study from [56] specified that *"Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them."* For instance, in a RL paradigm, an agent acts and received feedback in the form of rewards or penalties in order to learn how to complete a certain goal in an uncertain, possible complex environment [56].

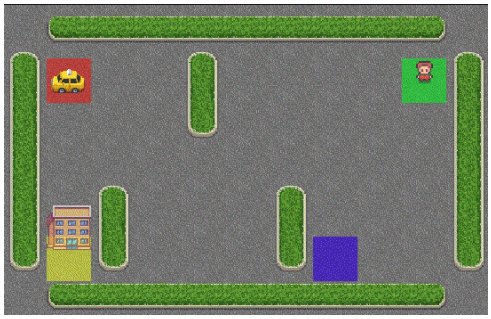

Figure 30: *Taxi-v3 environment* [28]

OpenAI offers different scenarios, and Taxi is being one of them. Thanks to the advancement of reinforcement learning (RL) this method is now evaluated and developed [26]. There are 2 ways to approach this Taxi-v3 environ-

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	Example: Object recognition,spam detetction

Figure 29: Difference between Reinforcement learning and Supervised learning [25]

ment using OpenAI Gym. A GitHub repository allows implementing the code and evaluating the various reinforcement learning algorithms [68].

4.3 Key Components

Deep learning (DL) together with machine learning (ML) have been tested to improve acceptance for decision-making of personalized electronics towards environmentally friendly production (carbon reduction emissions and improved resource material) [34]. The constant progression of machine learning has been collected significantly that challenges several models in manufacturing decision-making problems. The process of reinforcement learning has prevailed when it comes to reducing the necessity of training data. For instance, the system learns by itself over time with an actual operation [45].

4.3.1 Agent, environment, states, actions, and rewards

Reinforcement learning (RL) is a subset of machine learning discipline that is originally influenced by the biological system [56].In this paradigm, an **agent**, for instance, like human, robot, or vehicle, interacts along with the **environment**. Simultaneously, **states** and **rewards** are distinguished by the environment as the consequences of the action that is taken [45]. Figure below depicts the interaction of these key components.

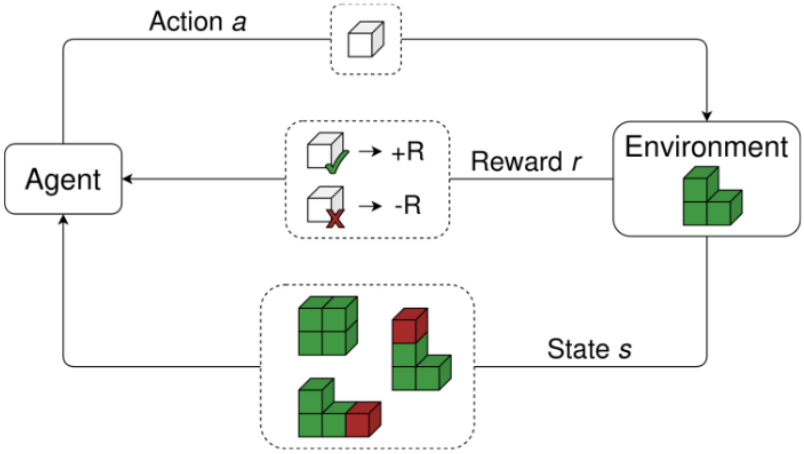

Figure 31: Agent and environment interaction. [56]

In simple terms, a state determines the current condition of the environment, where rewards are a buzzword concept that defines whether an agent is successful or fails. This kind of dynamic learning is pivotal because it entails an exploring solution that maximizes the entire benefits for the purpose of collecting relevance challenges that require decision-making [45]. Lastly is the component named **action**. It is a series of potential action A in a state S. In general, an agent's skills are similar across all the states. That is why, one set of A is presented [39].

4.3.2 Q-learning algorithm and its relevance to the problem

Q-learning is a part of reinforcement learning that explores an optimal strategy-selection rules for a finite Markov decision process (MDP). It assists an agent to adapt and optimize the total reward time after time via multiple contact with the environment, even if the model environment is unknown [54]. One of the off-policy strategies and maybe one of the frequently used in RL techniques is Q-learning. Since its inception, Q-learning has conducted countless and comprehensive research in RL and AI [39]. One of the open source repository by Efrann Panahi tested an overview of implementing Q-learning from the OpenAI Gym's Taxi-v3 environment. Check the image below

Overall, the implementation of the key components such as states, actions and rewards specified the environment setup where in an agent must be pick up and drop off customers by mapping a 5x5 grid. The Q-table and the Bellman equation [24] and the relationship between exploration and exploitation are the essential components where it was implemented to create an effective navigational tactic [48]. Autonomous maneuvering that is based on reinforcement learning and supervised learning in a clueless environment is essential. The synergy of these two learning tactics can be a very useful tool for handling unpredictable environment factors [4].

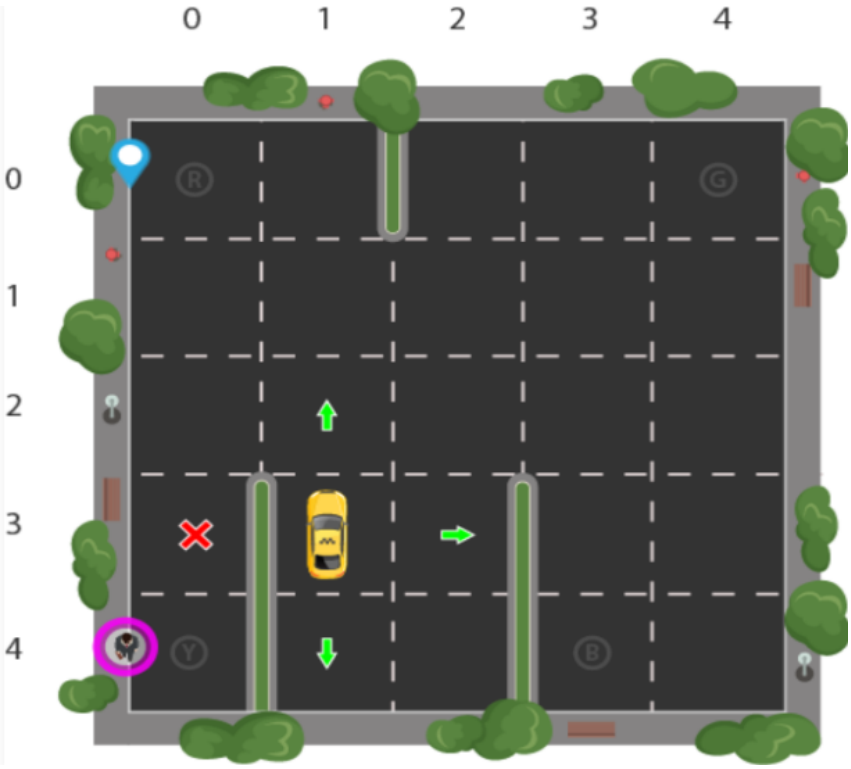

Figure 32: Q-learning implementation in OpenAI Gym's "Taxi-v3" environment [48]

4.4 Methodology

4.4.1 Q-Learning Overview

The reward table in Q-learning provides feedback that helps the agent (the driver) learn which actions are beneficial and which will yield negative results. This feedback comes in the form of rewards and penalties, guiding the agent's learning process. Each time the agent takes a particular action in a given state (for example, moving north, south, east, or west), the Taxi-v3 environment provides an immediate reward. These rewards can be either positive or negative.

As the agent receives rewards, it also updates the Q-values. The agent uses these rewards to adjust the Q-values stored in the Q-table, which records the estimated value of each action in each state. For each state-action pair, the agent calculates a new Q-value based on the received reward and its previous experience. Each Q-value tells you how "good" it is to do a certain thing when you are in a certain state.

By iterating over multiple episodes, the agent continuously updates the Q-values by exploring different actions and observing the outcomes. Over time, once the Q-values for all state-action pairs are sufficiently learned, the agent can choose the best actions based on the values in the Q-table. Q-values in the Q-table are mapped directly to both (state and action) combined.

The Q-value for a certain state and action pair shows how good the action is that can be done in that state. There is a greater chance of getting better awards when the Q-value is higher. if a taxi driver has already picked up a passenger and is at the destination, the Q-value for dropping the passenger off will probably be better than for any other action. Because of this, the agent's only choice is to drop off the passenger.

By default, Q-values are initially assigned arbitrary values since the agent hasn't explored the environment and has no prior knowledge of which actions will be beneficial. This approach allows the agent to start from a neutral position, giving equal consideration to all unknown actions. As the taxi driver continues to interact with the environment, it learns which actions lead to positive rewards and which lead to negative rewards, eventually improving the Q-values. The Q-values in the Taxi-v3 environment can be updated using the following equation [32].

4.4.2 Hyperparameters

α (alpha) - Learning Rate ($0 < \alpha \leq 1$): Controls the balance between newly learned information and previously learned information in the updated Q-value.

- $\alpha = 0$: The agent does not learn anything new and relies entirely on prior knowledge.
- $\alpha = 1$: The agent ignores prior knowledge and relies solely on the most recent information.

γ (gamma) - Discount Factor ($0 < \gamma \leq 1$): This value, between 0 and 1, represents the importance placed on future rewards.

- $\gamma = 0$: The taxi driver considers only the immediate rewards, ignoring future rewards.
- $\gamma = 1$: The taxi driver values both current and future rewards equally.

ϵ (epsilon) - Exploration Rate ($0 \leq \epsilon \leq 1$): Controls the agent's balance between exploration (trying new actions) and exploitation (choosing the known best action). Epsilon is used in epsilon-greedy policies, where the agent selects a random action with probability ϵ and the best-known action with probability $1 - \epsilon$.

- ϵ close to 1: The agent explores more, taking random actions more frequently to gather information about the environment. This high exploration rate is often used early in training to allow the agent to learn from a wider range of experiences.
- ϵ close to 0: The agent exploits more, relying on the learned Q-values to make decisions, which increases the likelihood of choosing the action with the highest estimated reward. This is typically preferred later in training once the agent has learned an effective policy.

Epsilon Decay: Over time, epsilon is often gradually reduced (epsilon decay) to shift the agent's focus from exploration to exploitation. This decay enables the agent to learn optimally while avoiding excessive random actions in the long run.
In updating the Q-value, the symbol $\leftarrow$ indicates that we are assigning a new value to the Q-value of a specific state-action pair. This new value is calculated by combining the old Q-value with the new information gained from the agent's recent experience.

- $(1-\alpha)Q$(state, action): This part represents the balance between old and new information. We include both the instant reward for the current action in the state and the highest reward the agent could get from the next state by adding the learned value.

Being able to change the Q-values in the Q-table over time helps the agent figure out which actions provide the best overall rewards. This allows the taxi driver to discover the optimal routes or sequences of actions that maximize total rewards.

Q-table: The Q-table stores the Q-values for each state-action pair. It serves as a map for the agent, helping it decide the best action to take in any given state based on accumulated experiences.

4.4.3 Algorithm Steps for Q-Learning

1. Setup: Initialize environment, Q-table, and hyperparameters. 2. Training Loop: For each episode, reset the environment. Choose actions, take steps, update Q-values, and track rewards and penalties. 3. After Training: Plot metrics and save the Q-table.

4.4.4 Deep Q-Networks (DQN) Overview

Traditional Q-learning relies on a Q-table to store values for each state-action pair, which becomes computationally impractical as the environment's state space grows. Managing a Q-table in such cases can lead to high memory and computational costs. DQNs address this challenge by employing a neural network to approximate Q-values, allowing the agent to generalize across similar states without explicitly storing every state-action pair. This generalization capability enables the agent to apply learned policies to new, unseen states, enhancing its adaptability and efficiency in large state spaces [65].

DQNs utilize experience replay, a method that stores past interactions (or "experiences") in a replay buffer. Instead of immediately learning from each experience, the agent randomly

samples from the buffer, which helps to break the correlation between consecutive experiences. This approach exposes the agent to a diverse set of experiences, fostering stable learning and improving generalization by allowing the agent to learn from a broader variety of state-action pairs. The reuse of experiences further accelerates convergence and enhances model reliability by reducing the risk of overfitting to recent states [51].

The target network is another key component in DQNs. DQNs employ two networks: the online network, which predicts Q-values and is updated continuously, and the target network, a delayed copy of the online network. This separation prevents rapid oscillations in Q-value estimates by stabilizing the learning process. The target network's periodic updates, guided by the Bellman equation, provide a steady reference point, improving convergence stability and allowing the agent to achieve near-optimal performance [9].

Neural networks in DQNs are inspired by the brain's structure, where interconnected neurons recognize patterns and adaptively learn complex functions. By leveraging neural networks in reinforcement learning, DQNs enable agents to solve complex, large-scale problems more efficiently than traditional Q-learning approaches [3].

Algorithms Setup

1.	**Initialize Environment and Agent:** Set up agent parameters (batch size, learning rate, epsilon, gamma, memory capacity).
2.	**Define Neural Network:** Create DQN model with embedding, hidden, and output layers.
3.	**Set Up Replay Memory:** Initialize memory buffer for storing (state, action, reward, next state).
4.	**Start Training Loop (per episode):** Reset environment, initialize rewards, calculate epsilon based on decay.
5.	**Select Initial Action:** Choose action using epsilon-greedy policy (random or highest Q-value).
6.	**Execute Actions and Update Q-Values:** Take action, observe reward, next state, done status. Store experience in memory. If enough memory, train model: Sample batch, compute Q-values and loss, backpropagate loss to update weights. Update state and action.
7.	**Update Target Network:** Periodically update target network to improve stability.
8.	**Save Model Checkpoints:** Periodically save model to Google Drive.
9.	**End Episode:** Record rewards and steps.
10.	**Plot Training Metrics:** Plot metrics after training completes.

Table 1: Q-Networks frameworks

Hyperparameters

Parameter	Description
Batch Size	Number of past actions used per learning step.
Gamma	Focus on future rewards (high) or now (low).
Epsilon	Chance of random actions. High = explore, low = use learned actions.
Eps Start	Starting randomness for exploring.
Eps End	Lowest randomness. Keeps some exploration.
Eps Decay	Speed of lowering randomness.
Target Update	Frequency of target network refresh.
Max Steps per Episode	Limit of moves per try.
Warmup Episode	Episodes of exploring before learning starts.
Save Frequency	How often to save progress.
Learning Rate (LR)	Speed of learning updates.
LR Min	Lowest learning speed allowed.
LR Decay	How quickly learning slows.
Memory Size	Amount of past actions saved.
Num Episodes	Total tries for learning.

Table 2: Hyperparameter Settings and Descriptions

4.4.5 State-Action-Reward-State-Action (SARSA) Overview

SARSA is an on-policy reinforcement learning algorithm, meaning it updates its Q-values based on the actions chosen by the agent's current policy for both the current and next actions. The SARSA algorithm follows a sequence of state-action pairs, as reflected in its name: State-Action-Reward-State-Action. After the agent performs an action in a given state, observes the resulting reward, and transitions to a new state, it selects the next action based on its current policy, often using an epsilon-greedy approach. SARSA updates the Q-value of the initial state-action pair using the following formula.

$$Q_{\text{new}}(S_t, A_t) \leftarrow (1 - \alpha) \cdot Q(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1})]$$

Where:

- S_t and A_t are the current state and action.
- R_{t+1} is the reward obtained after taking action A_t.
- S_{t+1} and A_{t+1} are the next state and action, selected based on the agent's policy.

- α (alpha) is the learning rate, and γ (gamma) is the discount factor for future rewards.

The target in SARSA includes the immediate reward plus the Q-value of the subsequent state-action pair, assuming the agent continues to follow its current policy. SARSA commonly employs an epsilon-greedy policy to manage exploration (trying new actions) and exploitation (choosing the action with the highest Q-value). Epsilon starts high to encourage exploration, allowing the agent to collect experiences across diverse states and actions. Gradually, epsilon decays, shifting the agent's behavior toward exploitation as it relies more on learned Q-values. In contrast to Q-learning, which is off-policy and updates Q-values based on the maximum future reward regardless of the current policy, SARSA is on-policy and updates the Q-values according to the actions selected by its own policy. This makes SARSA especially useful in situations where consistent policy behavior is advantageous [2].

4.4.6 Algorithms Steps

1.	**Initialize the Q-table:** Set up the Q-table with all zeros.
2.	**For each episode:** Reset the environment to get the starting state.
3.	**Choose an initial action:** Choose an action using epsilon-greedy policy (either a random action or the one with the highest Q-value).
4.	**While the episode is not done:** • Take the action and observe the reward and next state. • Choose the next action in the new state, again using epsilon-greedy. • Update the Q-table using the SARSA formula. • Set the new state and action as the current state and action.
5.	**Goal state reached:** If the goal state is reached, end the episode and move to the next one.
6.	**Reduce epsilon:** Gradually reduce epsilon to encourage more exploitation as training progresses.
7.	**Save the Q-table:** Save the Q-table after all episodes are completed.

Table 3: SARSA frameworks

4.5 Results - Q-Learning programs Evaluation

We have created two versions of a Q-Learning program: one with a fixed epsilon value and the other with an epsilon value starting at 1 and decaying over time. Both programs use the following hyperparameters: A) episodes = 25000, B) gamma = 0.99, C) alpha (learning rate) = 0.01, and D) epsilon = 0.5 for the first program; and A) episodes = 25000, B) gamma = 0.99, C) alpha (learning rate) = 0.01, and D) epsilon = 1 with decay over time for the second program.

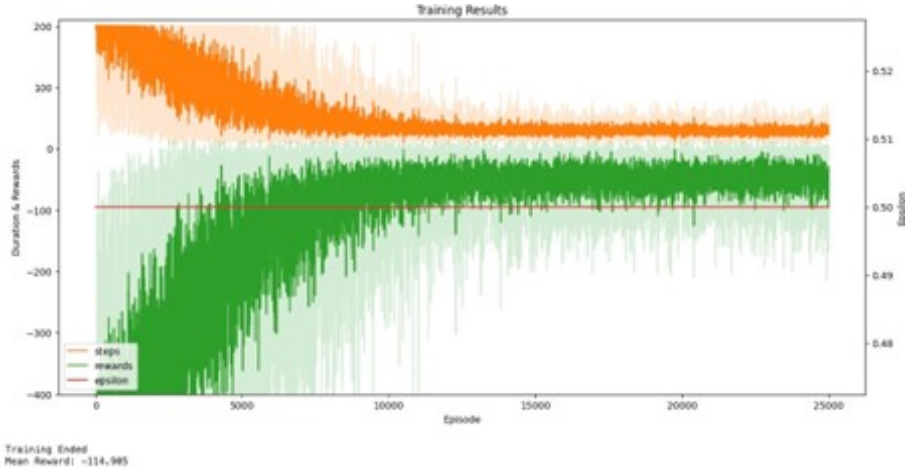

Figure 33: Average Result of our ACO Algorithm

Figure 33 In the graph, observe that the agent is actively exploring the environment, attempting to find an optimal solution in each episode. From episode 0 to around 10,000, our agent predominantly explores rather than exploits. During this period, the rewards fluctuate significantly, ranging from negative to positive values. This suggests that the agent has continuously explored throughout the training process instead of focusing on exploiting the learned information in the later stages. As a result, the agent may not have achieved convergence with a fixed epsilon value of 0.5, as it hasn't fully transitioned to exploitation.

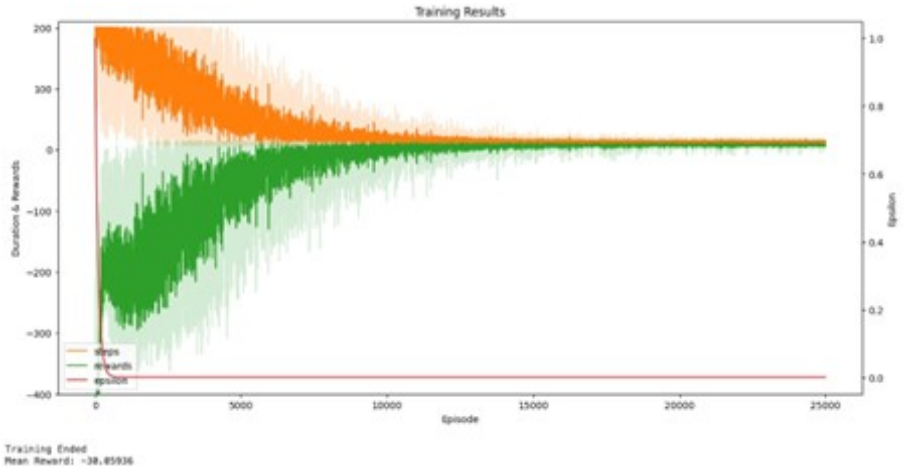

Figure 34: Average Result of our ACO Algorithm

Figure 34, in our graph, the agent begins by exploring extensively from episode 0 to around 10,000. Since we applied epsilon decay over time, the agent starts with a high epsilon value of 1, encouraging exploration. As epsilon decays, the agent gradually transitions from exploration to exploitation. This shift is evident as we observe that from around episode 15,000 onward, the agent requires fewer steps to complete each episode, and the rewards trend positively. This indicates that the agent is learning more effectively by focusing on actions that yield higher rewards. Consequently, we can conclude that the agent has likely achieved convergence in this program configuration, successfully balancing exploration and exploitation to optimize its performance. The program finishes at 42 seconds time.

4.5.1 Hyperparameters Tuning (Q-Learning)

Adjusting the hyperparameters to 'learning rate (lr) = 0.01' and 'gamma = 0.2' can lead to instability in the training process. In this case, the learning rate is set to 0.01, allowing for small updates to the agent's knowledge, while the gamma value of 0.2, or 20%, indicates that the agent will prioritize immediate rewards over future rewards. This low gamma value means that the agent will focus on short-term gains rather than the potential long-term benefits, which may lead to erratic learning and make it difficult for the agent to develop a stable strategy.

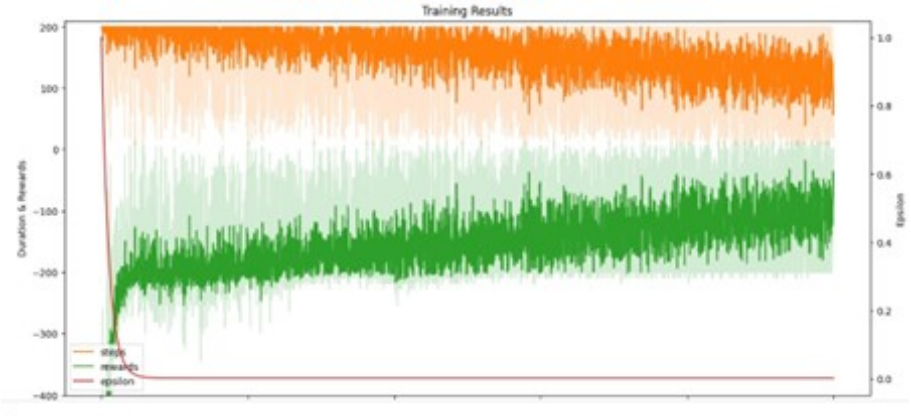

Figure 35: Average Result of our ACO Algorithm

Figure 35 depicts the graph demonstrating that the choice of hyperparameter values significantly impacts the stability of the Q-Learning training process. Inappropriate hyperparameter settings can lead to unstable training, causing fluctuations in the output from high negative penalty values to positive reward values. This variability highlights the importance of selecting appropriate hyperparameters to maintain consistent and effective training outcomes in Q-Learning.

4.5.2 Cumulative Rewards overtime evaluation for Q-Learning algorithm

Figure 36 illustrates the cumulative rewards over time for 100 episodes using the Q-Learning algorithm, suggesting that the agent consistently wins in each episode, accumulating positive rewards throughout. The hyperparameters used were: A) episodes = 2500, B) gamma = 0.99, C) alpha (learning rate) = 0.01, and D) epsilon = 0.5 for the first program; and A) episodes = 2500, B) gamma = 0.99, C) alpha (learning rate) = 0.01, and D) epsilon = 1 with decay over time for the second program.

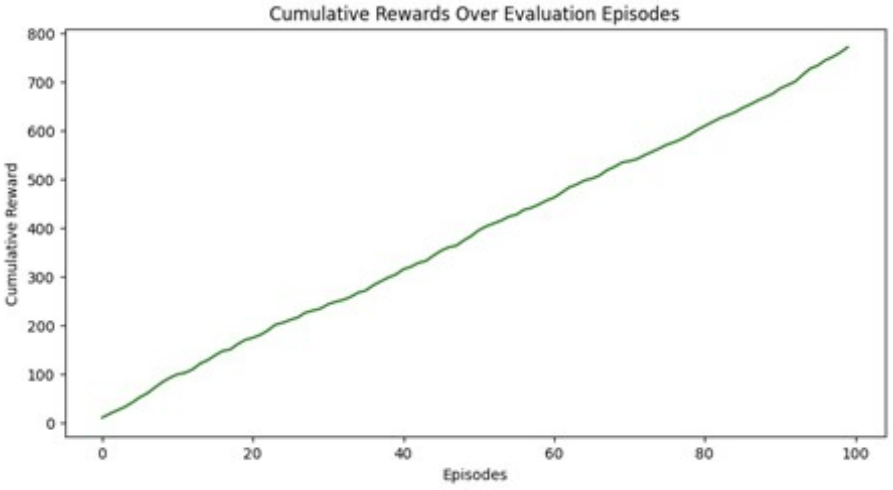

Figure 36: Rewards overtime evaluation for Q-Learning algorithm

4.5.3 Comparative Analysis of Q-Learning (with epsilon decay), Random Policy and Heuristic Policy

This table summarizes the performance metrics of three policies—Q-Learning, Random, and Heuristic—used for solving the reinforcement learning problem in the Taxi-v3 environment.

Policy	Avg Timesteps	Avg Penalties	Avg Reward	Win Rate
Q-Learning Policy	15.03	0.0	5.76	99.00%
Random Policy	193.96	63.17	-760.6	0.00%
Heuristic Policy	200.0	0.0	-200.0	0.00%

Average Timesteps: A lower value here indicates higher efficiency in reaching the goal, as the agent requires fewer steps to complete each episode.

Average Penalties: A lower number of penalties reflects the agent's ability to avoid mistakes. High penalties typically result from incorrect or inefficient actions.

Average Reward: A higher reward indicates better performance, as it reflects the agent's ability to complete the task effectively while maximizing positive outcomes.

The overall results show that the Q-Learning policy significantly outperforms the Random and Heuristic policies. Q-Learning achieves a high reward, incurs no penalties, requires fewer steps, and achieves a 99% win rate. This indicates that the Q-Learning agent has successfully learned an optimal strategy for the environment, consistently completing the task with minimal errors and high efficiency.

In contrast, the Random and Heuristic policies both fail to complete the task reliably. The Random Policy performs poorly, with high average time steps, a large number of penalties, a severely negative reward, and a 0% win rate, indicating that random actions do not effectively solve the task. The Heuristic Policy, while avoiding penalties, consistently fails to complete the task within the allotted steps, resulting in a negative reward and 0% win rate. This suggests that a simple rule-based strategy is insufficient to handle the complexity of the Taxi-v3 environment.

In summary, Q-Learning proves to be the most effective framework for this problem, as it optimally balances exploration and exploitation to maximize performance.

4.5.4 Deep Q-Networks

In our DQN implementation, we used the following hyperparameters: $\text{batch}_size = 128, gamma = 0.99, eps_start = 1.0, eps_end = 0.10, eps_decay = 400, target_update = 20, max_steps_per_episode = 100, warmup_episode = 10, save_freq = 1000, lr = 0.001, lr_min = 0.0001, lr_decay = 5000, memory_size = 50000, num_episodes = 10000, and architecture = 2.$

Figure 37: Average Result of our ACO Algorithm

Figure 37, suggests that our agent begins to explore more in the initial state between episodes 0 and 2000. However, the training process has stabilized from episode 2000 onwards, indicating that the agent achieves lower steps and positive rewards and has reached convergence in an early

state through epsilon decay over time. The training process ended after 39 minutes.

4.5.5 Hyperparameters Tuning (DQN)

Learning rate at 0.0001, the agent learns extremely slow that it takes countless hours to complete for 4000 episodes compared to 0.001 which usually finishes within 20 to 39 minutes.

Learning rate at 0.5, using this parameter, the training process just terminated after 4 hours with 236 episodes but excessive number of steps around 200,000 steps and same with negative rewards. It promoted instability of training the agent rather than helping the agent learn.

Training the models may take many hours and appears to be very sensitive to hyperparameters. It is also quite unpredictable in terms of training duration; the time to complete varies each time we run or execute the code for this policy. Sometimes, it completes in several minutes, while other times, it may take several hours, even causing Google Collab to timeout.

We also observed that during the initial stages of training, the model takes longer to execute each episode for the first 1,000 to 2,000 episodes. During this time, the agent is still in the exploration phase. However, as epsilon decays over time and reaches the hard limit of 0.10, processing speeds up per episode as the agent shifts from exploration to exploitation. The rewards also shift from fluctuating between negative and positive to mostly positive, and the number of steps required becomes lower at this point.

We have not changed the discount factor of 0.99, as it was the suggested value in various studies. This allows the agent to prioritize future rewards rather than immediate rewards.

4.5.6 SARSA Algorithm

In our training process, we will use the following hyperparameters: episodes set to 10,000, epsilon initialized at 1 as the starting exploration rate, max epsilon also at 1, gamma (discount factor) at 0.99, min epsilon at 0.001, epsilon decay at 0.01, and alpha (learning rate) set to 0.85. Our agent is configured with a gamma (discount factor) of 0.99, which means it has a strong focus on future rewards. This high gamma value allows the agent to prioritize long-term gains, factoring in future rewards when making decisions. Additionally, the learning rate (alpha) is set to 0.85, enabling the agent to learn rapidly while still giving considerable weight to immediate rewards. Together, these settings aim to balance learning from immediate experiences with planning for long-term success

Figure 38: Average Result of our ACO Algorithm

Figure 38 shows that during the initial stages of training, the agent received a high number of negative rewards and took many steps. Over time, however, the training process stabilized, with negative rewards and high step counts becoming less frequent. This indicates that the agent gradually learned to optimize its actions to achieve better outcomes. The entire training process completed within 11 seconds, demonstrating the efficiency of the training setup.

4.5.7 Hyperparameters Tuning (SARSA)

Adjusting the hyperparameters with the following values: episodes set to 10,000, epsilon starting at 1, $\max_e psilonat1, gammaat0.5, min_e psilonat0.001, epsilon_d ecayat0.01, andalphaat0.2$.
Here, gamma (0.5) indicates a balanced focus between future and immediate rewards, allowing the agent to weigh both near-term and long-term gains. The learning rate, alpha, is set to 0.2, meaning the agent will place some emphasis on experiences but not overly prioritize them, enabling more flexible adjustments based on new information.

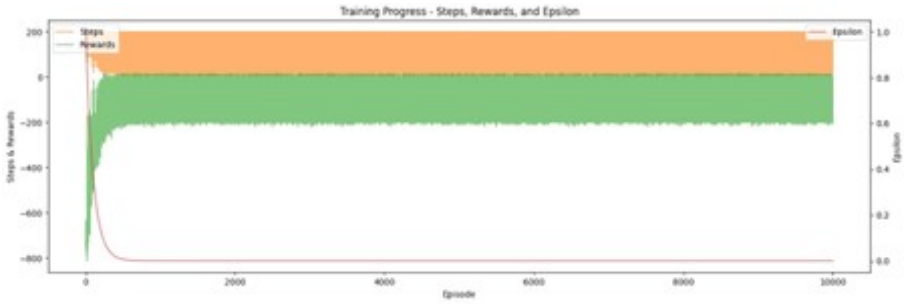

Figure 39: Average Result of our ACO Algorithm

Figure 39 illustrates that the agent consistently took numerous steps and received negative rewards throughout the episodes, indicating an unstable training process. This pattern suggests that the chosen hyperparameters were not effective in guiding the agent toward consistent learning. To achieve stability in the SARSA algorithm, an appropriate combination of hyperparameters is essential, as they directly influence the agent's ability to balance exploration and exploitation effectively and adapt to the environment over time.
Figure 40: Q-Learning requires a minimum of 25,000 episodes to achieve a stable training process,

	Q-Learning(Figure 2)	DQN	SARSA
Number of episodes	25000	10000	10000
Time finished	42 seconds	39 minutes	11 seconds

Figure 40: Comparative Analysis between Q-Learning, DQN, and SARSA

whereas both DQN and SARSA need only around 10,000 episodes to reach stability. DQN, with the right combination of hyperparameters, completes the training process in approximately 39 minutes, achieving stability efficiently. SARSA, on the other hand, is remarkably fast, finishing in under 11 seconds, followed closely by Q-Learning.

4.6 Discussion

4.6.1 Random Policy

The random policy allows the agent to take actions without any structured learning. The agent explores the Taxi-V2 environment by randomly selecting actions, which leads to a high degree of uncertainty in achieving task completion. Without a directed approach, the agent struggles to locate and transport the passenger effectively. As a result, convergence is not guaranteed, as the agent does not leverage any reward feedback or learning mechanism.

4.6.2 Heuristic Policy

A heuristic policy was implemented by directing the agent to move horizontally or vertically towards the passenger based on relative positioning. This approach performs well in environments without obstacles, as it enables efficient passenger pickup and drop-off. However, it is limited by its inability to navigate around obstacles. If a wall exists between the agent and the passenger, the agent cannot adjust its behavior effectively, leading to task failure in such cases.

4.6.3 SARSA(State-Action-Reward-State-Action)

SARSA demonstrated notable instability when hyperparameters were not well-tuned. Specifically, setting a discount factor (gamma) of 0.5 caused erratic training patterns, with the agent often accumulating high negative rewards. The learning rate (alpha) also had a significant impact; a high alpha value led to oscillations in the agent's behavior, while a low alpha slowed convergence. This underscores the importance of balancing gamma and alpha to account for

immediate versus future rewards. Achieving a stable learning process required careful management of exploration versus exploitation, emphasizing the need for a gradually decaying epsilon.

4.6.4 Q-Learning

The Q-Learning algorithm required extensive training, with around 25,000 episodes to reach stability, which made it comparatively slower than SARSA. A fixed epsilon led to prolonged exploration, hampering the agent's ability to converge. Only with epsilon decay did Q-Learning achieve an effective balance between exploration and exploitation, resulting in convergence. The epsilon-greedy strategy played a crucial role, and improper decay could lead to excessive exploration, undermining efficient learning. Hyperparameter tuning was essential for stability; a low gamma encouraged short-term actions, while high gamma values led to a more stable, long-term reward-based policy.

4.6.5 Deep Q-Networks (DQN)

DQN was highly sensitive to hyperparameters, especially the learning rate. A low learning rate, such as 0.0001, drastically slowed training, while a high learning rate (0.5) caused abrupt failures with excessive steps and negative rewards. Adjusting the learning rate was crucial, as a high alpha led to oscillatory behavior, while a low alpha slowed convergence. The training time of DQN was variable, sometimes causing Google Collab timeouts. Although epsilon decay improved efficiency over time, DQN's training required careful tuning to avoid instability and excessive time consumption.

4.7 Conclusion

In the Taxi-V2 environment, SARSA emerged as the most effective algorithm, providing a balance between stability and computational efficiency. Its relatively low training time and quick convergence make it well-suited for this task. Although Q-Learning and DQN showed promise, SARSA's ability to achieve stable performance with minimal computational resources positioned it as the optimal choice for this specific environment.

4.8 Future Works

In our current approach, hyperparameter tuning was primarily conducted through intuition, relying on guesses and random values. To enhance both the efficiency and stability of the training process, we recommend adopting systematic techniques such as Grid Search and Random Search.

These methods systematically evaluate all possible combinations of hyperparameters, offering a more structured approach that can lead to significant improvements in model performance [29].

Another promising avenue is to employ Reinforcement Learning for Hyperparameter Tuning. This approach enables a reinforcement learning agent to dynamically explore and select optimal hyperparameters based on performance metrics gathered from the training framework. By adaptively navigating the hyperparameter space, this technique can uncover better configurations, potentially surpassing traditional tuning methods in effectiveness and efficiency [50].

References

[1] Adams, J. L. (1961). *Remote control with long transmission delays*. Stanford University.

[2] Aljohani, T. M. and Mohammed, O. A. (2022). A real-time energy consumption minimization framework for electric vehicles routing optimization based on sarsa reinforcement learning. *Vehicles*, 4(4):1176–1194.

[3] Berezovsky, V. (2023). An overview of neural networks for medical image recognition. *E3S Web of Conferences*, 460:04028.

[4] Bin Issa, R., Das, M., Rahman, M. S., Barua, M., Rhaman, M. K., Ripon, K. S. N., and Alam, M. G. R. (2021). Double deep q-learning and faster r-cnn-based autonomous vehicle navigation and obstacle avoidance in dynamic environment. *Sensors*, 21(4):1468.

[5] Blasch, E., Pham, T., Chong, C.-Y., Koch, W., Leung, H., Braines, D., and Abdelzaher, T. (2021). Machine learning/artificial intelligence for sensor data fusion–opportunities and challenges. *IEEE Aerospace and Electronic Systems Magazine*, 36:80–93.

[6] Bourjot, C., Desor, D., and Chevrier, V. (2011). Stigmergy. *Self-organising Software: From Natural to Artificial Adaptation*, pages 123–138.

[7] Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, 99:300–313.

[8] Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

[9] Daley, B. and Amato, C. (2021). Human-level control without server-grade hardware. *arXiv preprint arXiv:2111.01264*.

[10] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91.

[11] Dixit, A., Kumar Chidambaram, R., and Allam, Z. (2021). Safety and risk analysis of autonomous vehicles using computer vision and neural networks. *Vehicles*, 3:595–617.

[12] Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*.

[13] Dorigo, M., Di Caro, G., and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172.

[14] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, 26(1):29–41.

[15] Duan, J., Zhu, Y.-a., and Huang, S. (2012). Stigmergy agent and swarm-intelligence-based multi-agent system. In *Proceedings of the 10th World Congress on Intelligent Control and Automation*, pages 720–724. IEEE.

[16] Earnest, L. (2012). Stanford cart. Accessed: 2022-02-27.

[17] Eiben, A. E. and Smith, J. E. (2015). *Introduction to evolutionary computing*. Springer.

[18] Elbeltagi, E., Hegazy, T., and Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced engineering informatics*, 19(1):43–53.

[19] Fagnant, D. and Kockelman, K. (2015). Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181.

[20] Feng, S., Sun, H., Yan, X., Zhu, H., Zou, Z., Shen, S., and Liu, H. (2023). Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615:620–627.

[21] Fiveable (n.d.). Ant colony optimization - (swarm intelligence and robotics) - vocab, definition, explanations. Accessed: 2024-09-26.

[22] Fonseca, C. M. and Fleming, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1):1–16.

[23] Gambardella, L. M., Taillard, , and Agazzi, G. (1999). Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill.

[24] GeeksforGeeks (2024a). Qlearning.

[25] GeeksforGeeks (2024b). What is reinforcement learning? Accessed: 2024-09-26.

[26] GoCoder (2024). Reinforcement learning tutorial with openai gym. Accessed: 2024-09-26.

[27] Gonsalves, T. and Upadhyay, J. (2021). Integrated deep learning for self-driving robotic cars. In *Artificial Intelligence for Future Generation Robotics*, pages 93–118. Elsevier, Amsterdam, The Netherlands.

[28] Gym Library (2024). Taxi-v3 environment — gym documentation. Accessed: 2024-09-26.

[29] Huang, J., Rojas, J., Zimmer, M., Wu, H., Guan, Y., and Weng, P. (2020). Hyperparameter auto-tuning in self-supervised robotic learning.

[30] Jabbar, A. M. (2018). Controlling the balance of exploration and exploitation in aco algorithm. *J. Univ. Babylon*, 26(4):10–17.

[31] Khayyam, H., Javadi, B., Jalili, M., and Jazar, R. (2020). Artificial intelligence and internet of things for autonomous vehicles. In *Nonlinear Approaches in Engineering Applications*, pages 39–68. Springer International Publishing, Cham, Switzerland.

[32] Kumara, V. (n.d.). Reinforcement learning with python: A guide to designing and solving problems.

[33] Lam, W. S., Lam, W. H., and Jaaman, S. H. (2021). Portfolio optimization with a mean–absolute deviation–entropy multi-objective model. *Entropy*, 23(10):1266.

[34] Leng, J., Ruan, G., Song, Y., Liu, Q., Fu, Y., Ding, K., and Chen, X. (2021). A loosely-coupled deep reinforcement learning approach for order acceptance decision of mass-individualized printed circuit board manufacturing in industry 4.0. *Journal of cleaner production*, 280:124405.

[35] Li, J. and Tan, Y. (2022). Information utilization ratio in heuristic optimization algorithms. In *International Conference on Sensing and Imaging*, pages 3–22. Springer.

[36] Li, S., Wei, Y., Liu, X., Zhu, H., and Yu, Z. (2022). A new fast ant colony optimization algorithm: the saltatory evolution ant colony optimization algorithm. *Mathematics*, 10(6):925.

[37] Li, Y. and Ibanez-Guzman, J. (2020). Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37:50–61.

[38] Lian, T. A. and Llave, M. R. (2014). Towards a multilevel ant colony optimization. Master's thesis, Universitetet i Agder; University of Agder.

[39] Littman, M. L. (2001). Value-function reinforcement learning in markov games. *Cognitive systems research*, 2(1):55–66.

[40] Liu, X., Chen, Y.-L., Por, L. Y., and Ku, C. S. (2023). A systematic literature review of vehicle routing problems with time windows. *Sustainability*, 15(15):12004.

[41] Mavrovouniotis, M. and Yang, S. (2013). Adapting the pheromone evaporation rate in dynamic routing problems. In *European Conference on the Applications of Evolutionary Computation*, pages 606–615. Springer.

[42] Mavrovouniotis, M. and Yang, S. (2014). Ant colony optimization with self-adaptive evaporation rate in dynamic environments. In *2014 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*, pages 47–54. IEEE.

[43] Metaheuristics.org (2024). Metaheuristics: From theory to applications. Accessed: 2024-09-26.

[44] Nahavandi, S., Alizadehsani, R., Nahavandi, D., Mohamed, S., Mohajer, N., Rokonuzzaman, M., and Hossain, I. (2022). A comprehensive review on autonomous navigation. *arXiv preprint arXiv:2212.12808*.

[45] Neves, M., Vieira, M., and Neto, P. (2021). A study on a q-learning algorithm application to a manufacturing assembly problem. *Journal of Manufacturing Systems*, 59:426–440.

[46] Nilsson, N. J. (1984). Shakey the robot: Technical note 323. *Artificial Intelligence Center, SRI International, Menlo Park, CA*.

[47] Ojha, V. K., Abraham, A., and Snášel, V. (2014). Aco for continuous function optimization: A performance analysis. In *2014 14th International Conference on Intelligent Systems Design and Applications*, pages 145–150. IEEE.

[48] Panahi, E. (n.d.). Taxi-v3-q-learning. GitHub repository.

[49] Patel, R. (2024). What is ant colony optimization (aco)? - a brief guide. Accessed: 2024-09-26.

[50] Rijsdijk, J., Wu, L., Perin, G., and Picek, S. (2021). Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 677–707.

[51] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[52] Schulze, J. and Fahle, T. (1999). A parallel algorithm for the vehicle routing problem with time window constraints. *annals of operations research*, 86(0):585–607.

[53] Sharma, P. and Gillanders, J. (2022). Cybersecurity and forensics in connected autonomous vehicles: A review of the state-of-the-art. *IEEE Access*, 10:108979–108996.

[54] Simplilearn (2024). Q-learning guide: Begin with reinforcement learning basics.

[55] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.

[56] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition.

[57] Tan, S.-Y. and Yeh, W.-C. (2021). The vehicle routing problem: State-of-the-art classification and review. *Applied Sciences*, 11(21):10295.

[58] Tavares, L. G., Lopes, H. S., and Lima, C. R. E. (2009). Construction and improvement heuristics applied to the capacitated vehicle routing problem. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 690–695. IEEE.

[59] Team, T. A. (2022). Ant colony optimization: An overview. Accessed: 2024-09-26.

[60] University of Magdeburg (2024). The traveling salesman problem. Accessed: 2024-09-26.

[61] Usmani, U. A., Happonen, A., and Watada, J. (2023). Revolutionizing transportation: Advancements in robot-assisted mobility systems. In *International Conference on ICT for Sustainable Development*, pages 603–619. Springer.

[62] Vala, T. M., Rajput, V. N., Geem, Z. W., Pandya, K. S., and Vora, S. C. (2021). Revisiting the performance of evolutionary algorithms. *Expert Systems with Applications*, 175:114819.

[63] Ventura, S., Luna, J. M., Ventura, S., and Luna, J. M. (2016). Introduction to evolutionary computation. *Pattern Mining with Evolutionary Algorithms*, pages 45–61.

[64] Wang, D., Tan, D., and Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing*, 22(2):387–408.

[65] Wang, K., Zhang, W., He, X., and Gao, S. (2017). Deep reinforcement learning of the model fusion with double q-learning. *DEStech Transactions on Computer Science and Engineering*, (aiea).

[66] Wikipedia (2024). File: Aco tsp.svg. Accessed: 2024-09-26.

[67] Zarrinpanjeh, N., Javan, F. D., Azadi, H., De Maeyer, P., and Witlox, F. (2020). Ant colony optimization parameter selection for shortest path problem. In *24th ISPRS Congress 2020*, pages 147–154. International Society for Photogrammetry and Remote Sensing (ISPRS).

[68] Zulfiqar, H. (2024). Reinforcement learning taxi problem. Accessed: 2024-09-26.

A Appendices

A.1 GitHub Repository Link

https://github.com/joimb9064/EAI_project_problems